Energy Conscious On-Chip Communication Bus Synthesis and Optimization for MPSoC Architectures

Vom Fachbereich 18
Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung der Würde eines
Doktor–Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

M.Sc.
Sujan Pandey
geboren in Kathmandu, Nepal

Referent:	Prof. Dr. Dr. h. c. mult. Manfred Glesner
Korreferent:	Prof. Dr. Jörg Henkel
Tag der Einreichung:	19. 12. 2006
Tag der mündlichen Prüfung:	15. 06. 2007

D17

Darmstädter Dissertationen
2007

Acknowledgments

There are so many people who acted as a source of help and inspiration during the almost four years of time and I owe a debt of thanks to all of them. In particular, I would like to express my sincere gratitude to the thesis adviser Prof. Manfred Glesner, who gave me an opportunity to explore a knowledge in his research institute and provided enormous supports and advises to materialize this work as a thesis.

I would also like to thank Prof. Jörg Henkel from University of Karlsruhe, Germany, for accepting as a reviewer of this thesis and giving me an opportunity to visit him to discuss about the content. His comments and remarks were valuable to improve the quality of this thesis and to envision future new research directions. Furthermore, I would like to extend my thanks to Prof. Dimitris Pavlidis, Prof. Udo Schwalke, and Prof. Jürgen Stenzel as members of examination committee. In this context, my sincere thanks to Prof. Schwalke for having a fruitful discussion on the technological aspects. His comments and remarks were also equally valuable to improve the quality of this work.

I had had lots of opportunity to interact and discuss with colleagues within the institute. Their constant supports were a result of my success in this carrier. Especially, I would like to thank Tudor A. Murgan and Leandro S. Indrusiak being good colleagues as well as friends. Further, I can not forget friendly colleagues Heiko Hinkelmann, Oliver Soffke, Oana Cobianu, Hao Wang, Petru Bacinschi, Massoud Momeni, Andre Guntoro, and Hans-Peter Keil. As part of the scientific management, I would like to extend my thanks to Thomas Hollstein, who helped me a lot from the moment I set my foot in Darmstadt to the end of my stay. In this regard, I also thank to Peter Zipf for his contribution from administration of GK to lectures and the scientific discussions.

I am equally indebted to the secretaries of institute, Silvia Hermann and Imgrid Wackermann, who helped me for many tiny problems. Further, without a well running system, I would not be able to carry out research and write my thesis. Thus, I would like to thank Andreas Schmidt for his valuable supports.

It is my such a pleasure to be around with good friends from school to university and I am very proud to have them. Especially, I thank to David Berner, not only for correcting my English, but also for giving me advices and suggestions when I was in need. I would also like to thank Binod Uprety for his constant inspiration. Furthermore, it

is difficult for me here to mention all good friends from my school, if their names are not listed, I assured that my gratitude is not less than for those listed below. Especially, I thank to Kishor Poudel, Lava P. Kuikel, Shiva C. Maharjan, and Bishnu Uprety with whom I shared all most my time. I would also like to thank Naresh Parajuli from Kathmandu University, who always encouraged me as a good friend.

At the end, I would like to express my sincere gratitude to my parents. The teaching and guidance of my parents and the constant encouragement of my brother Sajan are the main sources of inspiration. Here, I can not forget to mention my Manana "Kalu" for everything that she has devoted for me.

Sujan Pandey
18 July 2007, Bremen, Germany.

Kurzfassung

Heutzutage kann man in dem Entwurf moderner System-On-Chips zwei wesentliche Beobachtungen anstellen: Zum einen führt die zunehmende Systemkomplexität zu einem steilen Anstieg des Datenverkehrs der Busarchitekturen auf dem Chip. Zum anderen bewirkt die Technologieskalierung, dass Verbindungsleitungen immer dünner und somit Laufzeitverzögerungen immer größer werden. Diese beiden Nebeneffekte deuten darauf hin, dass der Entwurf von on-chip Datenbusarchitekturen auf dem Chip eine immer größer werdende Herausforderung für Systemdesigner wird. Das Ziel dieser Arbeit ist daher, Algorithmen zur Synthese von energieeffizienten on-chip Datenbussen zu entwickeln. Durch die Optimierung der Anzahl der Busse, der Busbreite und der Betriebsspannungen sind diese Algorithmen in der Lage, sowohl die Chipfläche als auch die Leistungsaufnahme der Datenbusse zu verringern.

Eine der Annahmen, die für die Synthese gemacht werden, ist, dass das betreffende System bereits vollständig partitioniert worden ist und diese Partitionen auf geeignete Module eines Multiprozessor System-on-Chips (MPSoC) abgbildet worden sind. Basierend auf diesen Modulen wird ein Task-Graph erstellt, der den Datenverkehr zwischen den on-chip Modulen modelliert. Die Problemformulierung des Syntheseverfahrens wird unterteilt in Scheduling, Allocation und Binding. Eine korrekte Formulierung dieser Probleme kann dann mit Hilfe von Optimierungswerkzeugen gelöst werden, welche die optimale Anzahl von Bussen und deren Breite bestimmen. Aufgrund der fortlaufenden Skalierung der Bauelemente und Verbindungsleitungen kann eine immer größere Anzahl an Transistoren auf dem Chip integriert werden. Dies führt zu einer Zunahme der Leistungsaufnahme pro Flächeneinheit, was wiederum eine verminderte Gerätezuverlässigkeit und Systemperformanz zur Folge hat. Es ist daher wesentlich, die Leistungsaufnahme während der Bussynthese zu berücksichtigen. Eine der Hauptbeiträge dieser Arbeit ist die Entwicklung eines Verfahrens, das eine gleichzeitige Datenbussynthese und Spannungsskalierung zulässt und dabei einen Kompromiss zwischen Kosten (d.h. Anzahl und Breite) für die Busstruktur und der Leistungsaufnahme eingeht. Die unbenutzte Zeit zwischen Kommunikationsaufgaben wird dabei genutzt, um den Bus zu teilen und die Betriebsspannungen herunterzuregeln. Da die Technik der kontinuierlichen Spannungsskalierung eine ideale Charakteristik für die Leistungsaufnahme erzeugt, kann sie nicht für den Dig-

italentwurf mit aufwendigen Spannungsreglern eingesetzt werden. Um dieses Problem zu umgehen, wird ein heuristisches Verfahren für die diskrete Spannungsregelung entwickelt, das in polynomialer Zeitkomplexität durchgeführt werden kann.

In einem echtzeit-eingebettetem System ist der zu übertragende Datenverkehr zwischen on-chip Modulen aufgrund der Vielfalt der Anwendungen nicht konstant. Des weiteren wird der Einfluss von Prozessparametervariationen auf die Systemperformanz mit zunehmender Technologieskalierung immer stärker. Um die Effekte, die von dem variablen Datenvolumen und der Prozessparametervariationen herrühren, zu integrieren, wird in dieser Arbeit ein erweitertes Verfahren für die Bussynthese vorgeschlagen. Das erweiterte Verfahren führt die gleichzeite Bussysnthese und Spannungsskalierung aus, allerdings unter Berücksichtung des variablen Datenvolumens und der zufälligen Prozessvariationen im worst-case Fall. Simulationen, die anhand von einem automatisch erzeugten Benchmark und einer realen Anwendung durchgeführt wurden, zeigen, dass eine intelligente Spannungsregelung während der Bussynthese sowohl die dynamische Leistungsaufnahme und die Leistungsaufnahme aufgrund von Leckströmen verringert als auch die Auswirkungen von Prozesstoleranzen mildert.

Abstract

Two major trends can be observed in modern system-on-chip design: first the growing trend in system complexity results in a sharp increase of communication traffic on the on-chip communication bus architectures. The second trend in technology scaling indicates that the wires are getting thinner and results in increment of wire delay. These trends, taken together, designing on-chip communication bus architectures is becoming an ever more challenging task for system designers. Thus, the aim of this thesis is to explore several algorithms that synthesize energy efficient on-chip communication buses. The algorithms reduce chip size and power consumption by optimizing the bus widths, the number of buses, and the voltage levels.

An assumption for synthesis is that a system has been partitioned and mapped onto the appropriate modules of a multiprocessor system-on-chip (MPSoC) architecture. Based on the partitioned and mapped modules, a communication task graph is extracted to model communication between on-chip communicating modules. The synthesis approach is formulated as scheduling, allocation, and binding problems. Once correctly formulated, these problems are solved with the help of an optimization tool to find the optimal bus width and the number of buses. As the device geometry and the wires are scaled down, a growing number of transistors can be integrated on a single chip, which leads to an increase in power consumption per unit area. This, in turn, results in the degradation of both device reliability and system performance. Thus, it is essential to optimize bus energy consumption during the synthesis of communication buses. As a major contribution, this thesis proposes a simultaneous on-chip communication bus synthesis and voltage scaling technique, that finds a trade-off between communication bus cost (bus width and number of buses) and energy consumption. The slack of each communication task is exploited in order to share communication bus usage and to scale down the bus operating voltages. As the continuous voltage scaling technique delivers an ideal energy consumption characteristics, it cannot be applied for the digital design due to the expensive voltage regulators. To cope with this problem, a heuristic for discrete voltage scaling technique is proposed, which can be solved in polynomial time complexity.

In a real-time embedded system, the amount of data to be transferred between on-chip modules is not fixed over time. This is due to the diversity of applications that

run on a single chip. Furthermore, as the process technology is scaled down, the effects of process variations are becoming a significant on system performance. In order to incorporate the combined effects of the data size and the process variations on the performance of communication buses, this thesis proposes an extended model for communication synthesis. The proposed model simultaneously performs on-chip communication bus synthesis and voltage scaling under data size and process variations. The problem is relaxed to a nonlinear optimization model, which synthesizes the optimal bus widths and the number of buses considering worst case data traffic and process variations. The experiments conducted on an automatically generated benchmark and real-life applications show that applying voltage scaling during the synthesis of on-chip communication buses effectively reduces dynamic power consumption, leakage power consumption, and mitigates the effects of process variations.

Table of Contents

List of Tables

List of Figures

List of Symbols

τ	Data processing task
c	Communication task
V_{dd}	Supply voltage
V_{bs}	Body bias voltage
V_{th}	Threshold voltage
NC_τ	Number of cycle
NB_c	Number of bit
dl	Deadline
w_τ	Data processing delay of task τ
b_r	Bus b with width r
$T_{s,c,r}$	Start time of task c
$T_{e,c,r}$	End time of task c
$\mathcal{N}_o$	Number of overlaps
$\mathcal{N}_\kappa$	Number of containments
R	Library of buses
d_o^{min}	Minimum overlap delay
T_d	Gate delay
$\epsilon_{i,j}^{\Delta V}$	Energy overhead due to voltage switching
$\delta_{i,j}^{\Delta V}$	Delay overhead due to voltage switching
C_{eff}	Effective capacitance
C_r	Capacitance of power rail
α	Technology scaling factor
ζ	Generalized scaling factor
κ	CMOS technological factor
c	Distributed ground capacitance per unit length
L	Length
ν	Speed of electromagnetic wave propagation
ρ	Resistivity of a conductor
ϵ	Permittivity of an insulator

H_ρ	Thickness of a metal conductor
H_ϵ	Dielectric thickness
$CLTI_{c,r}$	Data transfer delay of a task c with bus width r
$\lambda(l)$	Distribution function of wirelength
A_{comm}	Average number of communications
δ	Transition density of communication
S	Spatial correlation of communication
f	Frequency in Hz
$NB(\zeta)$	Random data size
μ	Mean
σ	Standard deviation
η	Timing yield constraint
$\phi^{-1}(\cdot)$	Inverse of an error function
$X_{c,t,r}$	A binary variable for a task c, at time t, and bus width r
$X_{c,t,r,V_{dd}}$	A binary variable for a task c, at time t, bus width r, and supply voltage V_{dd}
I	Identity matrix
T_{ox}	Thickness of oxide
E_{sat}	Electric field for the velocity saturation
α_{DIBL}	Drain induced barrier lowering constant

List of Abbreviations

TDMA	Time Division Multiple Access
CMOS	Complementary Metal Oxide Semiconducotor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
VLSI	Very Large Scale Integration
GALS	Globally Asynchronous Locally Synchronous
RTL	Register Transfer Level
TLM	Transaction Level Modeling
BCA	Bus Cycle Accurate
ILP	Integer Linear Programming
NLP	Nonlinear Programming
MILP	Mixed Integer Linear Programming
ASAP	As Soon As Possible
ALAP	As Late As Possible
CLTI	Communication Lifetime Interval
OCT	Overlap and Containment
CTS	Communication Tasks Scheduling
TS	Tabu Search
DVS	Dynamic Voltage Scaling
ABB	Adaptive Body Biasing
LTCT	Linear Time Cost Trade-off
DBS-CVS	Discrete Bus Width Selection and Continuous Voltage Selection
DBS-DVS	Discrete Bus Width Selection and Discrete Voltage Selection
HSB	High Speed Bus
LSB	Low Speed Bus
MC	Monte Carlo

Chapter 1

Introduction and Overview

Contents

The ability of the semiconductor industry to continually live up to Moore's prediction [110] has revolutionized the system-on-chip (SoC) design paradigm. With this paradigm, it is now possible to integrate multiple processing modules on a single chip in order to improve system performance, cost, power consumption, size, and time-to-market. To achieve all these goals, several aspects of system design methodologies have to be addressed including: first, it is important to have a proper specification language and simulation tools to model and simulate a system at system level. Second, it is also important to have an efficient partitioning of a complex system into hardware and software, mapping them onto a set of optimized on-chip modules like processors, DSPs, CPUs, etc. Third, it is also equally essential to have an efficient on-chip communication bus architecture to provide data transfer among on-chip modules. In this thesis, the main focus is the third part of the system design flow, which is the synthesis of on-chip communication buses.

The on-chip communication architecture is an interconnection network, which integrates multiple on-chip modules and provides data transfer among them. An obvious challenge of designing a communication bus architecture is to assure robust and efficient data transfer. This is essentially determined by the several factors including: 1) selection of an appropriate on-chip communication bus topology, 2) selection of the optimal bus widths, and 3) selection of a proper communication protocol. There exist several topologies, ranging from a single shared bus to more complex architectures,

such as hierarchical multiple shared buses, token rings, packet based communication topology with mesh structure, crossbars, etc. In a communication topology, we distinguish two types of on-chip communication modules in terms of their behavior: 1) master modules, (e.g., CPUs, DSPs, etc.), which are capable of initiating transactions and 2) slave modules (e.g., memories, peripherals, etc.), which respond to the transactions initiated by a master. As multiple masters often share a communication bus, communication protocols are used to avoid any conflict of bus access among the masters. The protocols define the bus arbitration policies such as round-robin access, TDMA, and priority based access. Furthermore, these protocols define the synchronization schemes and the burst size which determines for how many cycles a master is given a right without being required to request with the arbiter.

1.1 Motivation

In 1965 Gordon Moore wrote a famous article predicting the integration of 65,000 components on a single chip. At that time industry was capable to integrate only tens of transistors on a single silicon die. Even with this relatively small number of components, especially compared with today's complex system, pundits thought his predictions were exaggerated and probably optimistic [64]. Later, Robert Dennard developed a scaling theory showing that how Moore's law can be realized in practice. Since then the industry is following more or less[1] dutifully Moore's law for more than four decades. According to the 2005 international technology roadmap for semiconductors (ITRS'05) [10], experts are currently predicting that by year 2009 more than 4 billion transistors will be integrated on a single chip and it is expected that the number will increase further in the future. This is due to the advances in process technology and increasing demand of performance requirements for next generation multimedia, broadband, and network applications. Due to all of this we see more and more functionality being integrated onto a single chip which, in turn, has resulted in a sharp increase of overall on-chip communication traffic among the integrated modules. In such complex systems, on-chip communication is expected to become a major performance bottleneck [153].

To provide a mechanism to exchange data among multiple modules, the shared bus based architecture has been the most common choice for real-time distributed embedded systems. However, traditional architectures that are based on a single shared bus, often fail to satisfy stringent real-time performance requirements. Thus, trends in technology scaling and performance demand together, are driving us toward advanced SoC communication architectures, which range from multiple hierarchical buses to a network of buses.

[1] the original prediction of 12 months has been modified to 18 and 24 months

Despite the advantages of technology scaling trend to integrate more and more number of transistors for the higher integration, power consumption per unit area increases with shrinking device and wire sizes. As a consequence the device temperature increases, which, in turn, results reduction in mobility of carriers and degrades the speed of circuits. Recent data shows that more than 50% of all integrated circuits failures are related to thermal issues, which contribute to lower the semiconductor reliability. Thus, it is equally essential to optimize power consumption of on-chip communication buses during the sythesis.

Due to the diversity of applications to be run on a single real-time embedded system, the workload offered to it, is not uniform over time. This is why a communication bus architecture that has been synthesized without taking into account the system's peak load may turn into a major performance bottleneck. In the past, few efforts have been made to synthesize bus architectures for systems with a variable workload [96], where several applications are profiled at system level and the communication bus architecture is chosen for the worst case. Under normal load condition, however, the buses of such a system will be underutilized.

To come from the era of integrating tens of transistors to today's modern system-on-chip, which consists of about a billion of transistors, the road has not always been easy so far. The past leaps in chip integration have given various challenges such as yield, design productivity, lithography resolution, and power dissipation in their own time. There will be obviously more challenges to come, when the device feature sizes are getting scaled down to only few nanometers. Among them the most alarming challenge is the process variation. Recently, the 2005 international technology roadmap for semiconductors (ITRS'05) [10] has predicted that the process variation will cause critical challenges for manufacturability and yield. Its effect increases severely on the deep sub-micron technology as the feature sizes continue toward the sub-100 nanometer era. As a result of this, a circuit can have a completely different performance than expected. The variations occur because specific steps in the fabrication process, such as lithography, ion implantation, and chemical and mechanical polishing, are vulnerable to imperfections, noise, and imperfect control across time and locations.

Considering these trends, designing an efficient custom on-chip communication architecture to support the communication is a crucial problem to the system designers. The traditional on-chip communication architecture based on a single shared bus approaches are mostly based on the simulation of the entire system. However, the resulting architecture may not fulfill the requirements such as the performance, energy, size, etc. and the computational cost of simulation based techniques, which make these approaches infeasible when exploring a large design space. These issues have motivated the introduction of design automation tools that synthesize custom on-chip communication bus architectures.

1.2 Research Scope and Objectives

The aim of this thesis is to synthesize the optimal bus widths and the number of buses for real-time embedded systems. An assumption for synthesis is that a system has been partitioned and mapped onto the appropriate modules of an SoC and the synthesized on-chip modules exchange data through a shared bus. Based on the mapped on-chip modules, communication activities among them are extracted and formed a communication task graph, which consists of a set of communication tasks and their dependencies. The communication bus synthesis problem is formalized as an optimization problem, where communication tasks are scheduled for different bus widths in order to find the minimum communication cost. The resulting synthesis problem is solved using mathematical programming and a meta-heuristic algorithm (tabu search) to obtain a global optimal solution and a near-optimal solution, respectively.

As technology is scaled to sub-100 nanometer regime, power density has an increasing effect on the performance and reliability of an embedded system. This thesis proposes energy aware bus synthesis technique which performs a simultaneous on-chip communication bus synthesis and voltage scaling in order to find a trade-off between communication bus cost and energy consumption. The slack is exploited to share communication bus and to reduce energy consumption during the synthesis of communication bus. Further, in a real-time embedded system the amount of data to be transferred among on-chip modules is not uniform over time. This variability is modeled as a random variable and the synthesis problem is formulated as a nonlinear optimization problem, where bus synthesis and voltage scaling are performed simultaneously under variable data traffic. The approach synthesizes communication buses for the worst case scenarios.

According to the international technology roadmap for semiconductors (ITRS'05) [10] dealing with fluctuations and statistical process variations for a sub-nanometer scaled CMOS technology will be a challenging task and opens a lot of questions to designers. We propose an extended model, which combines the effects of data size and process variations on the performance of on-chip communication buses. The propose bus synthesis technique synthesizes energy efficient robust buses by mitigating the effects of process variations.

1.3 Thesis Outline

In a broad sense, this thesis is organized into three main parts: 1) preliminary, where technological related issues and previous work are discussed, 2) core part of the thesis, where the proposed approaches are presented, and 3) concluding remarks with

possible future research.

Preliminary: **Chap.** 2 and 3 are of introductory nature, defining the state-of-the-art techniques in bus synthesis. They are devoted to address different issues that effect the performance of on-chip communication buses. In particular, prior to communication modeling and synthesis, **Chap.** 2 gives a better understanding of different factors that influence the performance of on-chip communication buses. These factors are mainly classified into technology, layout, and architecture. As a system designer, it is essential to have knowledge about these effects at early design phases in order to model them at an abstract level. **Chap.** 3 discusses different state-of-the-art techniques in the synthesis of communication buses and gives a wide audience to our proposed techniques.

Core: **Chap.** 4, **Chap.** 5, and **Chap.** 6 represent the core part of this thesis. The goal of **Chap.** 4 is to synthesize the optimal bus width and the number of buses assuming that a system has been partitioned into Hw/Sw and mapped onto the appropriate modules of an SoC. Based on this model, a communication task graph is extracted to model on-chip communication behavior. The bus synthesis problem is formulated as scheduling, allocation, and binding problems. As a scheduling problem, first communication tasks are scheduled using mathematical programming to obtain the global optimal solution. Since the algorithmic complexity for global optimal solution is **NP**-hard, later a heuristic based on tabu search is proposed to find a near-optimal solution. As an allocation-binding problem, a clique partitioning algorithm is used to find the number of buses. Further, a communication bus refinement technique is proposed based on swapping and moving on-chip modules from one bus to another. The objective is to minimize data transfer through a bridge in order to reduce delay and power overhead. **Chap.** 5 proposes an energy efficient communication bus synthesis technique where voltage is scaled to minimize energy consumption. The algorithm performs simultaneous bus synthesis and voltage scaling, where the slack of each communication task is exploited to share the bus and to scale down the voltage. Thus the resulting synthesis problem is an optimization problem that finds a trade-off between communication bus cost and energy consumption. **Chap.** 6 presents an approach to synthesize energy efficient communication buses under data size and process variations. The effect of variations is mitigated by simultaneously performing communication bus synthesis and voltage scaling during the synthesis. The variation in data size is due to the diversity of applications to be run on a single embedded system. However, process variations are due the parameters such as channel length, width, threshold voltage, etc. The proposed technique models the effects at system level and synthesizes robust communication buses. In **Chap.** 7 the bus synthesis methodology is validated by conducting experiments

on several benchmarks.

Conclusions: Finally, **Chap.** 8 gives a conclusion of this thesis and shows some possible future research directions.

Chapter 2

Influential Factors to the Performance of On-Chip Communication Bus

Contents

While designing a complex system with about a billion transistors, there is a wide range of factors such as technology, layout, system complexity, etc. that affect the performance of a system in terms of power, delay, circuit reliability, time-to-market, etc. If one of the design decision is taken wrong, this will lead us to a bad design and results loss of revenue. Thus, it is essential to consider those factors early in a design flow so that their effects can be modeled at higher levels of abstraction and designers are able to take the right decision at the right time. As the main topic of this thesis is

the synthesis of on-chip communication buses, we consider five different factors that affect the performance of on-chip communication buses. These are technology scaling factors, layout related factors, architecture, system complexity, and algorithms. Technology and layout related factors can be characterized with passive elements of electric circuits such as resistances, capacitances, and inductors, which influence the delay, power consumption, and circuit reliability. In our communication bus synthesis algorithm, we model these factors and explore their possible impact on communication buses.

This chapter is organized as follows. **Sec.** 2.1 describes how the technology scaling trend influences the performance of wires. It further shows how process variation can have a significant impact on the performance of communication buses. **Sec.** 2.2 addresses the layout related issues such as proper interconnect planning, the combination of retiming and the partitioning of circuits, buffer insertion, and wire width planning so that the performance of post synthesis communication buses can be improved. **Sec.** 2.3 addresses mainly architectural issues that have to be fixed before the synthesis of communication bus. These are communication bus topologies, bridges, routers, switches, synchronous, and asynchronous communications. Finally, **Sec.** 2.4 gives the summary of this chapter.

2.1 Technology Scaling Trends

Technological factors refer to all the physical parameters that appear after a system has been mapped onto a target CMOS technology. As a result of this, the parameters include parasitic components such as resistance, capacitance, and inductance which differ for different technology nodes. In this subsection, we show how technological related factors influence the performance of an on-chip communication bus architecture and how these factors can be modeled at system level so that a designer can take a decision in an early design phase. Our main focuses are first to analyze the effect of device and wire scaling and second, the effect of process variations on the on-chip communication bus.

2.1.1 Device and Wire Scaling

The principle of the MOSFET device scaling theory is based on a *constant electric field* as proposed by Dennard et al. [51]. The main idea behind the device scaling technique is to increase the performance in terms of delay and power consumption and to reduce the device geometries, while still preserving the basic operational characteristics of MOSFET devices. When all of the voltages and dimensions such as channel length,

	Constant Field Scaling	*Generalized Scaling*
Channel Length	$1/\alpha$	$1/\alpha$
Channel Width	$1/\alpha$	$1/\alpha$
Gate-Oxide Thickness	$1/\alpha$	$1/\alpha$
Electric Field	1	ς
Voltage	$1/\alpha$	ς/α
Doping	α	$\varsigma\alpha$
Gate Delay	$1/\alpha$	$1/\alpha$
Power Dissipation	$1/\alpha^2$	ς^2/α^2
Power Density	1	ς^2

Tab. 2.1: Constant electric field and generalized device scaling [154]

channel width, wiring width, and insulator thickness are reduced by the scaling factor α and the doping and charge densities are increased by the same factor, the electric field inside the MOSFET remains as it was in the larger device. This is called constant electric field scaling and results increase in circuit frequency and integration density in proportion to the factor α and α^2, respectively. These constant electric field scaling relations are summarized in column two of Tab. 2.1 for some of the important physical parameters. However, there are two main problems of this constant electric field scaling method. First, the built-in potentials cannot be scaled due to its strong tie with silicon band gap energy, which does not change unless a different semiconductor material is used. Furthermore, since the subthreshold slope is determined by the thermodynamics of the Boltzmann distribution of carriers, it cannot be scaled either. Consequently, the threshold voltage cannot be scaled by much. The result is an exponential increase in leakage current, which appears to be a dominant factor at the sub-nano meter scaled CMOS technology nodes. Fig. 2.1 depicts actual past and projected future scaling of voltage and oxide thickness versus channel length. It can be seen that because of above mentioned limitations, voltages cannot be scaled at the same rate as the channel length. While in earlier generations of MOSFETs, voltages were scaled linearly, which resulted in an increase of carrier velocity and device performance. However, recently, carrier velocity of devices has been saturated, but voltage scaling has been slow because of nonscaling of the subthreshold slope and the leakage current.

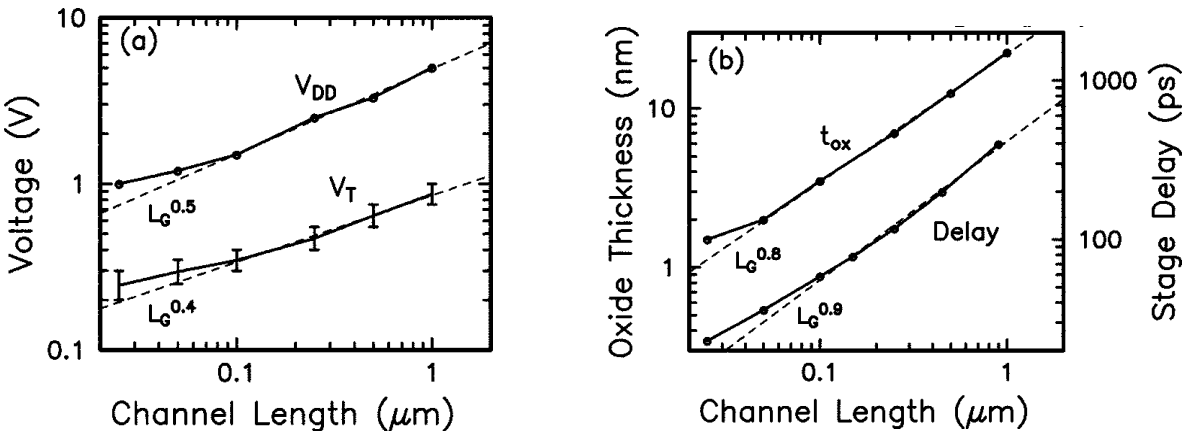
The constant electric field method appears therefore to be impractical to keep device scaling trend. This can be accommodated by introducing an additional scaling factor ς, which allows to increase the electric field with factor ς. This method is called generalized scaling rules as summarized in column three of Tab. 2.1. However, it offers

two main disadvantages, which are the reliability issue of devices due to the increase in the electric field and an increase in total power consumption per area due to the cramming more number of components onto the integrated circuit. This introduces other challenging issues such as device packaging and cooling requirements. As the temperature increases, the mobility of carriers decreases and this results in low device gain. This, in turn, would reduce the speed of the circuit.

Further the scaling trend shows that signal wires and devices should be scaled with the same scaling factor in order to increase integration density. In this case, if the number of devices, which are to be integrated on a chip is constant then the overall delay of remains almost same due to the reduction in mean wire length. However, this is not the case for each technology node, where more and more transistors are integrated on a single chip. This results in an increase of global wire length and resistivity, since the resistance is proportional to the wire length and inverse of cross-section. Furthermore, for each scaled technology node, the number of metal layers increases to enhance connectivity among the devices. To cope with such an ever increasing wiring problem, wires can be classified into three different types: local, intermediate, and global wires [154]. The local and intermediate wires are used mainly to route local signals between gates within a larger block of gates. These wires are typically short so that the total delay due to *RC* (resistance and capacitance), remains almost the same despite the shrinking in cross-section of wire. However, the current density of local wires increases and this results in problems such as device reliability due to electromigration. The influence of this problem can be minimized by scaling the wires with factor $\sqrt{\alpha}$ instead of α as shown in Tab. 2.1.1.

The second major concern of scaled local wires is an effect of coupling capacitances due to the shrinking trend of pitches, i.e., the distance between two wires. The consequence is higher crosstalk noise, which degrades the signal integrity and modifies the power consumption of the wire line drivers. Since local wires are typically very short, the effects of self and mutual inductance are negligible, and thus, local wires can be characterized by only the coupling capacitance. The coupling capacitances can be minimized by using a wire with an aspect ratio of about 2 [154], which minimizes the distance between wires. This can be achieved by reducing the width of the wires and keeping the height almost constant. However, this is still a trade-off between resistance and noise.

In contrast to local wires, global wires are longer and they are used for power grids, clock tree distribution, global data buses, and other important signals on the top layers of metal. Unlike in local wires, the ideal and quasi-ideal scaling technique can not be applied to global wires. Since, global wires connect blocks within a chip, its average length increases by factor $\sqrt{\alpha}$ if the wire is scaled by factor α. So, to allow unattenuated signal transmission along the global wires with low resistance, a constant dimension



(a) (b)

Fig. 2.1: Past and projected future scaling trends for CMOS logic. (a) Supply voltage and threshold voltage versus channel length. (b) Gate oxide thickness and 2-in NAND delay versus channel length [155].

	Local Wiring		*Global Wiring*	
	Ideal Scaling	*Quasi-ideal Scaling*	*Ideal Scaling*	*Constant Dimensions*
Wire Width	$1/\alpha$	$1/\alpha$	$1/\alpha$	1
Wire Thickness	$1/\alpha$	$1/\sqrt{\alpha}$	$1/\alpha$	1
Wire Length	$1/\alpha$	$1/\alpha$	$\sqrt{\alpha}$	$\sqrt{\alpha}$
Resistance	α^2	$\alpha^{3/2}$	α^2	1
Capacitance	1	≈ 1	1	1
RC Delay	1	$1/\sqrt{\alpha}$	α^3	α
Current Density	α	$\sqrt{\alpha}$	α	$1/\alpha$

Tab. 2.2: Wire scaling scenarios for local and global wires [154]

scaling technique is used to keep the wire resistance constant. This approach follows the concept of fat wires suggested by [139].

2.1.2 Effects of Process Variations

Variations are deviations in the value of process parameters and they collectively affect the performance characteristics of a circuit from their given specification. There are mainly two sources of variation: process variations and environmental variations. The process variations are due to the manufacturing process such as change in effective

channel length, channel width, oxide thickness, etc. The environmental variations are due to the environmental factors such as temperature, pressure, and humidity, which are independent from the manufacturing process. In the past deterministic worst case models were used to model the effect of variation on the system, however, they are overly pessimistic for the sub-nano meter scaled CMOS technology node and may lead to increased design effort and longer time-to-market, which ultimately may result in lost revenues. To overcome the above problems, a new statistical timing analysis method [147, 121, 103] is being used widely in academia and industry. Before we start with an impact of variation on performance parameters, we introduce simple delay models for device and wire. The alpha power delay model of a device can be written as

$$T_{device} = \kappa \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{2.1}$$

where, V_{dd} is supply voltage, V_{th} is threshold voltage, κ and α are technology dependent parameters. Similarly, a simple delay model for the wires is given by the distributed *RC* delay (assuming that $rcL^2 > L/\nu$) as,

$$T_{wire} = rcL^2 \tag{2.2}$$

where, r is distributed resistance per unit length, c distributed ground capacitance per unit length, L interconnect length and ν is speed of electromagnetic wave propagation. If we consider a simple parallel plate model for the parasitic capacitance per unit length of the interconnect, the interconnect delay of Eq. (2.2) can be re-written as,

$$T_{wire} = \frac{\rho\epsilon}{H_{\rho}H_{\epsilon}}L^2 \tag{2.3}$$

where, ρ is the resistivity of the conductor, ϵ is the permittivity of the insulator, H_{ρ} is the thickness of the metal conductor, and H_{ϵ} is dielectric thickness. From Eqs. (2.3) and (2.1), overall performance of the wires depends on the gate delay (which acts as a driver of a wire) and the delay of the wire itself. Since the technological parameters in the above equations are not deterministic due to the process variations, there is a strong influence of process variation on the performance of the on-chip communication bus. For instance variation in gate length is among the most critical parameter; it has a significant effect on both inter-die variation (resulting from variation in duration of exposure) and intra-die variation (resulting from lens aberration and other lithography effects) [140, 173]. In Eq. (2.1) the terms κ and V_{th} depend on the number of process parameters such as channel doping concentration, channel length, channel width, oxide thickness, and supply voltage due to short channel effect. The gate delay is inversely proportional to the square of threshold voltage V_{th}, which depends on several process variation parameters. If there is a slight variation of threshold voltage from the nominal value, it can be vulnerable to the performance of the on-chip communication bus. Similarly, an increase in variability of interconnect parameters such as wire

width, wire thickness, wire height, and resistivity ρ affect the wire delay. The result shows that variations in gate-length are expected to increase significantly as compared to other process parameters, with variability increasing in all parameters [115].

Recently, adaptive body biasing technique [40,159,27,116] has been shown to be an effective method of post-silicon tuning of a circuit to reduce variability under the presence of process variations. The basic principle is to manipulate the transistor threshold voltage, V_{th}, through the body effect, which provides either a forward or a reverse body effect to change threshold voltage. This principle can be explained using drain current conductance in terms of a series of resistances between source and drain. This resistance increases (decreases) with the increase (decrease) in body bias voltage, V_{bs}, which results in change in transistor performance in terms of delay and power consumption. Although, the body biasing method is a proper way to mitigate the effect of process variations, it adds complexity on the design tools and the distribution network can be expensive in terms of silicon area.

2.2 Layout Related Factors

The continuous scaling of feature sizes in semiconductor technologies has opened a new era so-called sub-100 nanometer scaled CMOS technology, which can integrate about 4 billion transistors on a single chip with an operating frequency of 12 to 13 GHz in the 20nm technology by year 2009 as projected in the 2005 international technology roadmap for semiconductors (ITRS'05) [10]. Furthermore, with increasing feature size scaling, it is expected that the interconnects will play a dominant role on the performance of circuit instead of devices. In past, significant amount of work has been done in the area of on-chip communication bus optimization at different levels of abstraction, however, most of these still follow the conventional VLSI design flow with emphasis on design and optimization of logic and devices. Interconnect optimization typically is done either by layout designers or automatic place and route tools. After the completion of layout if there is a timing violation due to long wires, designers have to iterate the whole process, which effects time-to-market and revenue. This motivates the interconnect-centric design flow, which integrates interconnect planning at early design stages which has tremendous impact on the final result. Fig. 2.2 depicts a model of wire, which is characterized by its physical and electrical parameters. These parameters are extracted after the floorplanning and routing of a circuit on the two dimensional layout. Fig. 2.2(a) shows a wire with multiple capacitance at equal distances, while Fig. 2.2(b) gives a simplified wire model with equivalent capacitances at the source side and the load side. The corresponding source and load capacitances of

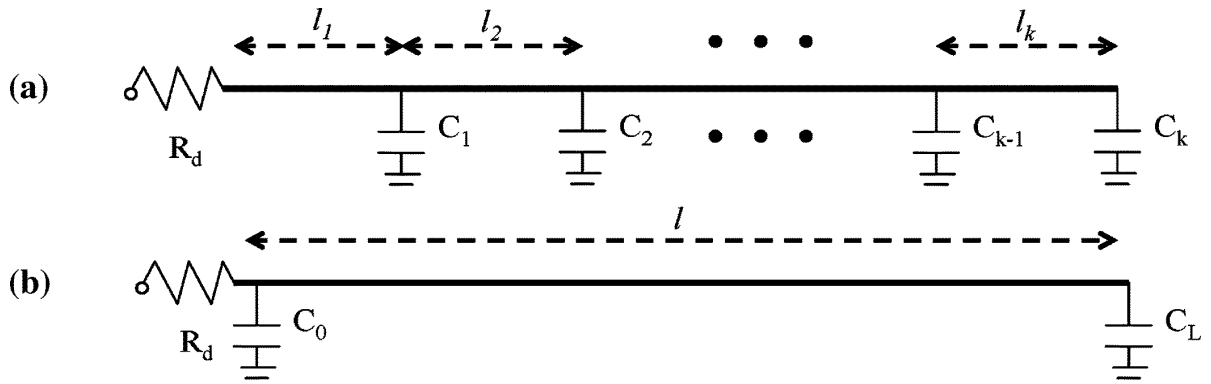


Fig. 2.2: Wire delay model [44],. (a) Multiple pin net. (b) Multiple two pin net

a wire can be calculated as [44],

$$C_o = \sum_{j=1}^{k} C_j - C_L \tag{2.4}$$

$$C_L = \sum_{j=1}^{k} \frac{\sum_{i=1}^{j} l_i}{l} \cdot C_j \tag{2.5}$$

Once the floorplanning and routing have been done, the total length of wire can be estimated and its corresponding delay is given as [44],

$$T_{int} = R_d C_o + \left(\frac{\alpha_1 l}{W^2 \alpha_2 l} + \frac{2\alpha_1 l}{W \alpha_2 l} + R_d c_f + \sqrt{R_d r c_a c_f l} \right) \cdot l \tag{2.6}$$

where $\alpha_1 = (1/4) r c_a$, $\alpha_2 = (1/2)\sqrt{r c_a / R_d C_L}$, and $W(x)$ is Lambert's W function defined as the value of w which satisfies $we^w = x$. The term R_d is the resistance of a driver, and l is the length of a wire. Furthermore, r is the sheet resistance in Ω/sq, c_a is unit area capacitance in $fF/\mu m^2$, and c_f is unit fringing capacitance in $fF/\mu m$ (defined to be the sum of fringing and coupling capacitances) and rest other terms in Eq. (2.6) are technology dependent parameters. After a circuit has been mapped onto the layout using floorplanning and routing, electrical parameters such as resistances and capacitances cannot be optimized that much enough due to their dependency on the technology as shown in Eq. (2.6). However, the physical parameters of a wire, which act as a dominant performance factor, can be optimized through different algorithms and methodologies. The algorithms can be characterized into two classes: in terms of type of solution they find, which are globally optimizing solutions and locally optimizing solutions. For globally optimizing solutions, the algorithm searches all possible floorplannings and routings of a system. It then determines the optimal wire length by minimizing or maximizing a given objective function subject to a set of constraints. As

they explore all possible implementations, the complexity of these algorithms can be exponential or **NP**-hard, which is one of the main disadvantage of the global optimal solution. Then again for a local optimal solution, the algorithm searches possible implementations within a local region and returns a locally optimal cost of the objective function in a polynomial time complexity. This later turns out to be a fast method to find a solution, however, it does not give the best solution. Hence, it is a trade-off between time complexity and the quality of solution.

Furthermore, independent of the type of solution, the layout and interconnect optimization depend on the way an objective function is defined. For instance, it can be optimized for power, delay, size or their combination and their results will be different for different objective functions. The conventional flooplan and routing approaches [124,168] minimize the objective function, which includes the total chip area, subject to a set of delay constraints. However, these techniques cannot be used anymore for today's systems with about half a billion transistors and wires; the resulting solution can be vulnerable for the power consumption and the maximum operating frequency due to a long critical path. Recently, in [187] a floorplan method is presented with a combined cost function of chip area and power consumption. The results show that paying the penalty for a small increase in area can reduce power consumption significantly while still respecting the given required arrival time of the signals.

In addition to above methods, there are several wire delay optimization techniques, which can be applied simultaneously with floorplanning. The most common that are being used by industries and academia are retiming, buffer insertion, and wire width planning. In the following subsection the details of these wire delay optimization techniques are discussed.

2.2.1 Interconnect Planning

Since the complexity of the wiring problem is increasing, the planning of interconnects at the early design stages is essential to generate an efficient layout with the minimum wire delay. In general, interconnect planning can be divided into three main steps, which are physical hierarchy generation, floorplanning with interconnect planning, and interconnect architecture planning [44]. After these steps, the resulting floorplan is evaluated under a set of constraints and if it satisfies the constraints the solution will be accepted. Otherwise several iterations will be carried out until the solution meets all constraints.

The main problem in a system design process is to map different hierarchies of interconnects and modules at system level description onto a two-dimensional layout with little or no consideration of the layout information. Although the high level hardware description languages facilitate the hierarchical design techniques at each level

of abstraction to reflect the logical dependency and relationship of various functions and components in the design, it is still impossible to find the best mapping from logic hierarchy into physical hierarchy. This is due to the gap between logic and physical hierarchy. This gap can be filled by generating a good physical hierarchy of interconnects. As the first step of interconnect planning, the physical hierarchy generation step partitions all interconnects into a set of different physical hierarchies before floorplanning. These include global, semi-global, and local interconnects, which are generated by using a classical mincut algorithm. It partitions a system into a set of main blocks and each block is further partitioned into sub blocks. The communication between two main blocks is classified as a global interconnect, while the communication between sub blocks within a block is characterized as local interconnects. Based on the physical hierarchies of interconnects further steps such as floorplanning and routing can be conducted efficiently and the results show significant improvement of the layout in terms of wire delay, size, and power consumption. The second step of interconnect planning is called physical level interconnect planning. This interacts with the interconnect synthesis tools and plans for the best interconnect topology, wire width, wire ordering, wire spacing, etc. for global, semi-global, and local interconnects. The third step is the interconnect architecture planning, which exploits the degree of freedom provided by the process technology and identifies the technology dependent parameters that influence the overall system performance, reliability, and power consumption subject to the manufacturing constraint. The parameters include the number of routing layers, the thickness of each interconnect, the thickness of the isolation layer, the metal resistivity, each layer's dielectric constant, each layer's nominal width and spacing etc. Each of the above parameters is adjusted to optimize the layout in terms of the target clock rate, interconnect distribution, depths of the logic network, etc.

2.2.2 Combine Retiming and Partitioning

After the planning of the interconnects, placement and routing of the circuit are performed simultaneously to find the minimum chip area. In this step, a large circuit is partitioned into a set of blocks and sub blocks using a mincut algorithm, which finds the best partition of circuits in terms of minimum number of cutsize, i.e., the number of wires connecting the circuits. However, while minimizing the objective function of cutsize, the algorithm may find partitions with a long critical path connecting two circuits. The delay of the critical path can be optimized using retiming simultaneously with a partitioning algorithm [44]. Retiming is an optimization technique, which improves the speed of a synchronous circuit by relocating registers without changing the circuit functionality [98]. Fig. 2.3 depicts a motivational example for simultaneous retiming and partitioning of a circuit. Assume that each node has delay 1, the inter-block connection delay is 2 and intra-block connection delay is 0. In Fig. 2.3(a), before applying

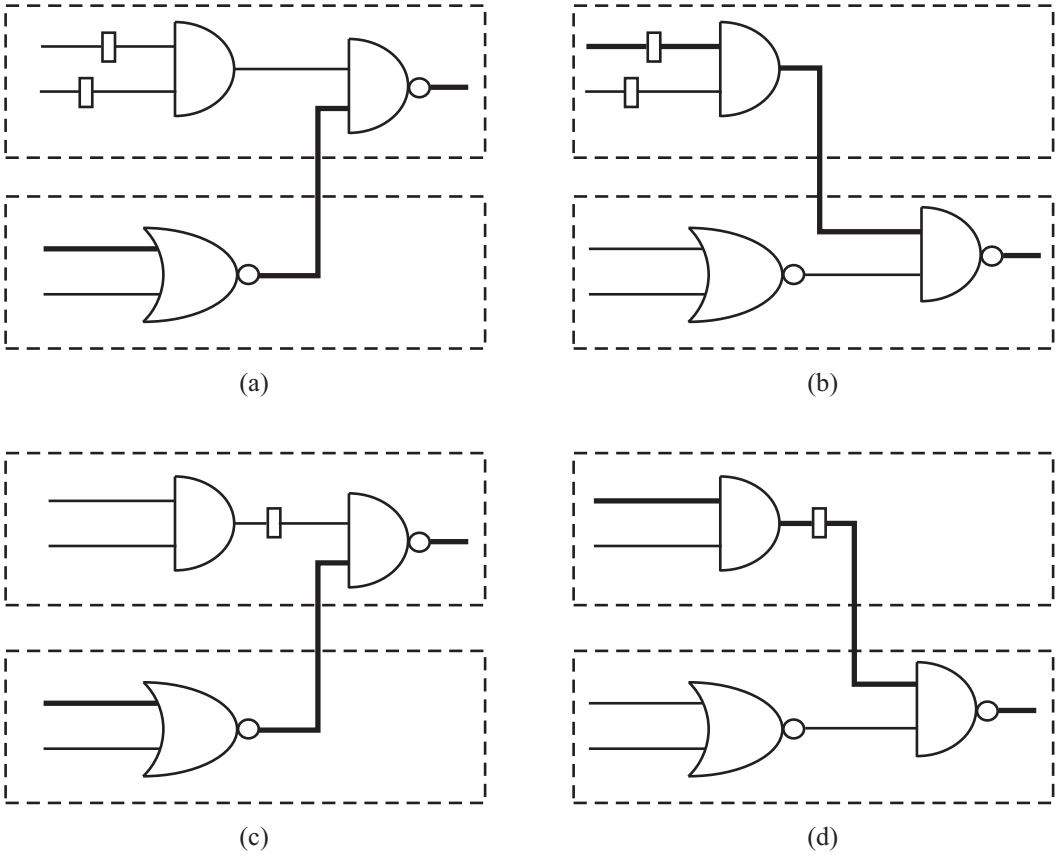


Fig. 2.3: Simultaneous partitioning and retiming for wire delay minimization [44]. (a) Cutsize = 1 and delay of critical path = 4. (b) Cutsize = 1 and delay of critical path = 4. (c) Cutsize = 1 and delay of critical path = 4. (d) Cutsize = 1 and delay of critical path = 3.

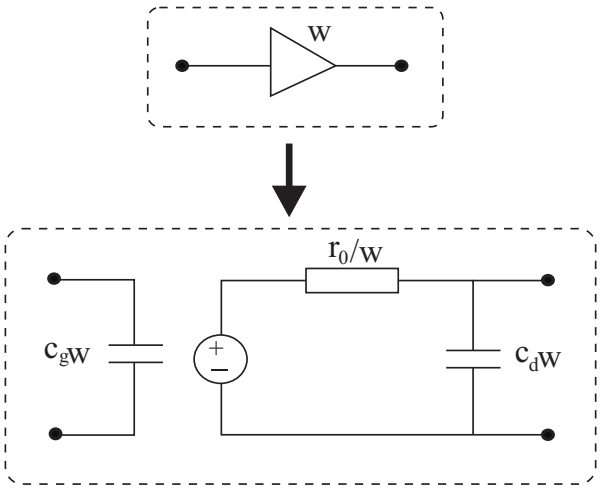


Fig. 2.4: Small signal model of Buffer [44]. The buffer size is given by w; r_0 is the output resistance of a unit-sized buffer; c_g and c_d are the gate and drain capacitance.

retiming, number of cutsize is 1 and delay of critical path is 4. After applying retiming in Fig. 2.3(c), there is no change in the number of cutsize nor in the critical path delay. While for another partition of circuits shown in Fig. 2.3(b), the retiming results the number of cutsize same as before, however, the delay of the critical path is improved from 4 to 3 in Fig. 2.3(d). This example clearly demonstrates that the simultaneous retiming and partitioning method may improve the critical path delay significantly after just relocating existing registers within a circuit.

2.2.3 Buffer Insertion and Wire Width Planning

In addition to above discussed optimization techniques, the buffer insertion technique is also applied most commonly to improve the performance of the communication bus during floorplanning and routing [164,102,15,44]. The basic idea behind this technique is to find the longest wire, that violates the timing constraint and insert buffers in between nodes connecting circuits. It optimizes delay mainly in two ways, first partitioning a long wire into small pieces of wire and second decoupling off-path capacitances, which make the buffer insertion technique one of a powerful tool to optimize *RC* delay. Fig. 2.4 depicts a buffer model and its equivalent small signal model, where the size of buffer, its equivalent series resistance, and capacitances at gate and drain terminal are characterized by the channel width w. Since the resistance of a wire is proportional to its length, the buffer insertion improves the wire delay. Similarly, due to the decoupling effect of a buffer, it decouples the capacitances of divided wire segments and improves the overall delay as shown in Fig. 2.4. As the technology scaling trend is growing exponentially, the total number of buffers required is increasing with decreasing transistor feature size. It has been shown that the number of buffers required for

the 70nm technology node is close to 800,000 [44] and this is expected to grow for the coming generation. Although the buffer insertion technique seems like a promising candidate to improve the interconnect delay, it offers some challenges to a designer in terms of increased power consumption and increased area of the layout. Thus, an efficient algorithm is required to perform buffer planning so as to minimize the power consumption and area of layout, while still meeting the delay constraint. Buffer insertion techniques, which have been carried out in past, can be categorized as pre-layout insertion and post-layout insertion. In post-layout insertion, topological information of interconnects can be utilized for timing analysis and it appears to be practical to optimize the delay. While for pre-layout buffer insertion, there is no information about the layout and interconnect plan, so the algorithm performs buffer insertion, floorplaning and routing simultaneously. As a result, the complexity of the appropriate algorithm is **NP**-hard [158]. However, the pre-layout buffer insertion technique can give a better optimized solution in terms of the number of buffers, layout area, and routing of interconnects. Early works on post-layout buffer insertion techniques are presented in [164, 102] with polynomial and pseudo-polynomial run-time complexity, respectively.

Although the buffer insertion technique is promising for long wire delay optimization, it has some limitations such as power consumption and obstacles for buffer insertion due to existing circuit blocks in the layout, which make buffer insertion impractical even if there is enough space available to optimize the wire delay. In presence of these limitations, another technique so-called wire width planning can be used to enhance high speed data transfer between on-chip communicating modules. Wire width planning also called wire sizing is an effective technique to reduce the delay of interconnects, however, the floorplanning and routing process can turn out to be complicated in presence of a set of different wire widths. Ideally, the continuous wire sizing technique gives the global optimal solution, but it may not be applicable due to its practical limitations in the manufacturing process. Furthermore, since the wire resistance is inversely proportional to its width, its geometry cannot be increased arbitrarily because of given area constraints. Hence, the wire sizing technique is a trade-off between the wire performance and the area of the layout. For a given layout of a design, the wire sizing problem can be defined as [44],

$$\Phi(\vec{W}, l_{min}, l_{max}) = \int_{l_{min}}^{l_{max}} \lambda(l) \cdot f(\vec{W}, l) dl \tag{2.7}$$

where $\lambda(l)$ is the distribution function of wirelength l, l_{min}, and l_{max} are the minimum and maximum wirelengths for this metal layer, $f(\vec{W}, l)$ is the objective function to be minimized by the design and $\vec{W}$ is the wire width vector. In Eq. (2.7), the complexity of the optimization problem depends on the number of discrete wire widths. Early work of Cong et al. [43, 42] proposed an $O(n^r)$ wire sizing algorithm for an n segment tree with r possible wire widths. Their objective function is a linear combination of the sink

delays. Later Sapatnekar presented an improved wire sizing algorithm, which find the best upper bound of the objective function with run-time complexity $O(r^n)$ [141]. In [102] Lillis et al. presented a pseudo-polynomial time algorithm that performs buffer insertion and wire sizing simultaneously to find the best trade-off between them.

2.3 Architecture

In the above subsection, we discussed effects of technological and layout related factors on the performance of on-chip communication buses. To over come the problems of shrinking wire width and an increase in length, several state-of-the-art techniques were pointed out to enhance the performance of wires. In this section, we focus more on the architectural issues such as the selection of appropriate on-chip communication bus topologies, communication protocols, methods of message transformation, etc. Traditionally, single shared on-chip communication bus architectures are classified according to the operating mode e.g., synchronous or asynchronous and arbitration schemes e.g., centralized, decentralized, or distributed. However, they may not meet the demand, as feature sizes become smaller and the cross-sectional area of wires decreases, causing wire resistance to increase and signal delay to grow. In the era of about a billion-transistors architectures, signals do not reach across the chip within one or two clock cycles. The estimated results show that less than 1% of a chip is reachable in a single clock cycle [13]. Then again, with the increasing trend in system complexity, there is a huge demand of communication placed by on-chip communication traffic, on the on-chip communication architecture. To cope with the ever increasing problems of technology scaling and system complexity, communication architectures can be classified into four major classes based on their network topology [53]. These are shared-medium networks, direct networks, indirect networks, and hybrid networks.

In a shared-medium based on-chip communication architecture, the transmission medium is shared by several on-chip modules and it is used most commonly in an embedded system due to its simplicity. An alternative to this topology is a dedicated point-to-point connection between two neighboring on-chip communicating modules. Communication between any two neighboring modules takes place via a point-to-point connection, while the communication between any two non-neighboring modules takes place through intermediate modules. This type of network topology is called direct communication network. Instead of using intermediate modules, communication between any non-neighboring modules can be achieved by means of one or more switches. This type of network is called indirect communication network. In an embedded system, all communicating modules may not need available bandwidth provided a given on-chip communication topology and this results in an under utilization or over utilization of the given communication architecture. A hybrid communica-

tion network, however, can be used to utilize the communication resources more effectively. As the on-chip communication architecture plays an important role on the performance of embedded systems, above communication networks address several design factors such as performance requirements, scalability, incremental expandability, and reliability.

2.3.1 Communication Topologies

Shared-Medium Network: This network has the lowest communication architecture complexity, where the communication media (usually a bus) is shared by several communicating modules and data transfer between them takes place in a time multiplex manner. Each module attached to a shared communication media has a standard interface to transmit and receive the data. There are mainly two types of standard shared-medium networks: which are shared-medium local area networks and shared-medium backplane bus. The local area network is mostly used to interconnect computers that span physical distances to few kilometers. Contrarily, the backplane buses are mainly used for on-chip communication in multi-processors embedded systems. As all modules share a single communication media, more than one module may access it at the same time, which results in a media access conflict. A arbitration strategy is therefore an important issue to determine the mastership of the shared-medium to resolve conflicts. Due to performance and implementation reasons, it is impractical to have a centralized control or to have some major fixed access assignment to determine the bus master. Thus, the local area network uses distributed media control, which can be classified as contention bus, token bus, and token ring. While the backplane bus uses a centralized media access controller, a so-called arbiter that grants a permission as a response to the bus request from a communicating module to take the control over a bus. Immediately after the bus is granted, the bus master puts informations such as address and data on the backplane bus. After the successful completion of a data transfer between modules, the bus master releases its control over the bus. There are mainly two ways to release the bus: release-when-done and release-on-request. The first one releases the bus when a data transfer is completed; this is called centralized arbitration. The second one holds the bus until another processor request it. This type of arbitration technique is called distributed arbitration.

Direct Network: A direct communication network consists of a set of nodes, where each node has a direct point-to-point connection with another node in the network. Each node is programmable with its own processor, local memory, and other functional unit. Fig. 2.5 depicts a generic architecture of a node. The nodes within a network can have different functionality such as storage, DSP processor, vector processor,

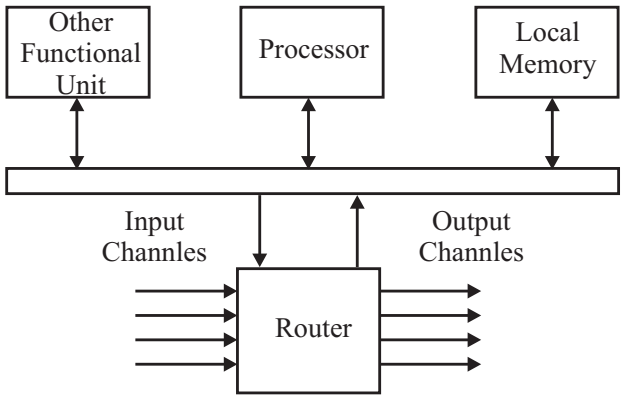


Fig. 2.5: A generic architecture of a node in a direct network [53]

etc. One common component of all nodes is the router, which routes data from one node to another node using a point-to-point connection. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of a system also increased. Since scalability is a key issue in designing multiprocessor embedded systems, direct networks have been a popular interconnection architectures for designing large-scale parallel processing systems. In comparison to this, bus based communication architectures are not scalable as they can become the bottleneck when more processors need to be integrated in an existing architecture. A direct network can be characterized by three main factors: topology, routing, and switching. A topology defines how nodes are interconnected with each other. Ideally, topology can be a completely connected graph nodes, where all nodes are connected by dedicated point-to-point connections. In this case, there is no need to use intermediate nodes to transfer data from one node to another, and the result is a fast interconnect network. However, this topology can not be realized in a practice due to several limitations such as wiring area, wiring cost, and its complexity. The most popular direct communication network is the n-dimensional mesh, and the k-ary n-cube or torus, where all nodes are not fully connected. For these networks, an intelligent routing algorithm is needed to route a message from source to the destination through intermediate nodes. When a message reaches an intermediate node, a switching mechanism determines how and when a message has to be routed toward the destination.

Indirect Network: Unlike in direct networks, instead of having a direct point-to-point connection between two nodes, indirect networks consist of a set of switches that route messages from source to destination. Each node has a network adapter that connects to a network switch, which can have a set of input and output ports. Similarly to the direct network, indirect networks can be classified by three factors which are network topologies, routing, and switching. Its topology is defined by the interconnections

between switches. In an ideal indirect network, all nodes are connected by a single $N \times N$ switch, which is called crossbar. The crossbar network provides an interconnection for any processor in the system to any other processor or memory unit so that many processors can communicate simultaneously without any access conflict. When more than one processor try to access different memories simultaneously, the arbitration allows them to access all of them without conflict. However, when more than one processor try to access the same memory unit there will be a conflict and the arbitration lets one processor proceed while the others wait. The arbiter in a crossbar network is distributed among all the switches. In addition to the crossbar network, multistage interconnection networks (MIN) and generalized MINs model are also most commonly used in parallel computing.

Hybrid Network: It is a combination of a shared-medium, direct networks, and indirect networks. There are several types of hybrid networks: multiple backplane buses, hierarchical networks, cluster based networks, etc. One approach to increase the bandwidth of shared-backplane buses is to have a multiple buses, however, due to the limitation of electrical packaging technology, multiple buses are used very rarely in practice. Another way to increase the bandwidth of networks is to have hierarchical buses with a global bus at the top, which are connected by either bridges or routers. In this case, again the global bus may become a bottleneck. Another common type of hybrid network is the cluster-based network, which is very similar to the hierarchical network. Instead of a global bus at the top of the hierarchy, it has point-to-point connections between nodes, just as like in a direct network.

2.3.2 Bridges, Routers, and Switches

Bridges, routers and switches are used to facilitate the message transfer between communicating modules. In general bridges are used to connect more than two buses, with identical or different communication protocols. When the protocols are different for different buses, interfaces for each bridge should be different and this appears to be challenging in terms of design perspective.

In direct networks routers are used to route messages from source to destination. They read the headers of incoming messages and find the shortest possible path to forward them. Especially in n-dimensional mesh architectures, routing algorithms can be complex with increasing mesh sizes.

As soon as a channel is selected by a routing algorithm, switches connect the input port to an output port. There are mainly two types of switching: circuit switching and packet switching. In circuit switching, a complete connection between source and the destination is established first and then the transfer of message takes place. In packet

switching the message is transferred as soon as the channel is reserved. Thus packet switching appears to be more effective in terms of resource utilization compared to circuit switching.

2.3.3 Globally Asynchronous and Locally Synchronous

As feature sizes shrink and the die sizes increase, synchronization of future system-on-chip design with a single global clock and with negligible clock skew is becoming a major challenge for the silicon technology. To cope with this problem, a new paradigm called globally asynchronous and locally synchronous (GALS) architecture is proposed in [28, 171, 113]. The architecture consists several synchronous on-chip modules with an asynchronous wrapper around of it. The main idea is to partition a system into several clock islands that communicate with each other in a self-timed fashion. Thus the functionality of each module can be described and synthesized with a well established synchronous design flow. Based on this concept of GALS architectures, Benini et al. proposed a new SoC design paradigm called Networks on Chips (NoC) [22]. It is an interconnection network for high-performance parallel computers with multiple processor and memory blocks. The aim is to solve future SoC architectural and design productivity issues by providing a uniform communication network connecting multiple modules and standardizing the handling of various inter-module communications. Furthermore, NoC architectures provide re-usability of existing intellectual property blocks, physical-architectural-level design integration, and platform-based design methodologies.

In contrast to the GALS architectures, in [41] a synthesis for on-chip multicycle communication architecture is presented for a synchronous design. Their focus is on the synchronous designs and propose a way to systematically handle multicycle on-chip communication. The technique is based on regular distributed register (RDR) microarchitecture, which offers high regularity and direct support of multicycle on-chip communication.

2.4 Summary

In this chapter, we have discussed different factors such as technological, layout related, and architectural which affect the performance of on-chip communication buses. On one hand, there is the increasing trend in device and wire scaling, which enables to integrate an increasing number of transistors on a single chip. However, on the other hand, this trend has a significant negative impact on wire delay, power consumption per area, and other parameters. As a result, system performance and reliability will be

degraded. Thus, these factors have to be modeled at higher levels of abstraction and different optimization techniques have to be applied to mitigate their effects. Among them supply and body bias voltage scaling, bus encoding, buffer insertion, wire width planning, etc. are commonly used techniques to improve circuit performance. Furthermore, independent of technological factors, the selection of an adequate architecture and a good architecture optimization algorithm has also an impact on the performance of the communication bus. For instance a global search algorithm can find the best solution, however, in general, it can not be applied in practice due to its complexity in terms of time and space. There are some other algorithms that are called heuristic, to find a near-optimal solution of a problem in a polynomial time complexity. Thus, it is a trade-off between run time complexity and the quality of solution.

Chapter 3

State-of-the-Art in Communication Bus Synthesis and Optimization

Contents

In the previous chapter we have discussed several issues that influence the performance of on-chip communication buses. Among them technological and layout related issues have an impact on the passive elements such as resistance, capacitance, and inductance of a circuit. These elements degrade the circuit performance in terms of power consumption, delay, reliability, etc. Similarly, other issues such as the selection of an adequate architecture has also an impact on the throughput of the communication bus. In this chapter we present different existing optimization techniques to refine

a communication bus architecture in presence of different influential factors as previously discussed. These techniques are categorized on the basis of level of abstraction on which the respective refinement techniques are applied.

This chapter is organized as follows. **Sec.** 3.1 illustrates the importance of transaction level communication modeling including bus interface synthesis, the Khan process based communication model, and the abstract channel model. The main goal of modeling at the transaction level is to explore different possible implementations of the communication bus architecture such as the selection of protocols, network topologies, etc. **Sec.** 3.2 presents bus cycle accurate level synthesis techniques such as real-time constraint driven, layout aware and floorplan aware communication bus synthesis. This technique lies one level of abstraction below transaction level modeling and includes several implementation details of a system. Thus, it is slower in terms of run time than transaction level modeling. **Sec.** 3.3 presents different power and delay optimization techniques for a synthesized system. Finally, **Sec.** 3.4 gives a summary of this chapter.

3.1 Transaction Level Communication Modeling

System-on-chip designers are facing design challenges due to the ever increasing system complexity. Today SoC designs of a complex system have several multiple IPs (CPUs, DSPs, FPGAs, memories, peripherals, etc.), which communicate with each other by exchanging data through system buses. In such a complex system, on-chip communication becomes a major performance bottleneck [153]. Although on-chip communication bus architectures such as OCP [7], AMBA [1,55] and CoreConnect [4] have been popular choices in current designs of SoCs, they have opened up a large exploration space because they can be configured in so many different ways [126]. Thus, system designers have to explore a large design space to find an efficient communication bus architecture with the optimal bus width, the number of buses, and the best communication protocol. Traditionally, systems were captured at a cycle and pin-accurate level in register transfer level (RTL) and then simulated for performance estimation before synthesis. This is, however, practically impossible for today's large and complex systems, as it would require tremendous amount of memory and processing power. This has motivated a new paradigm of modeling a complex system at an abstract level, where an early estimation of a system characteristics can be done before committing to the RTL development.

Transaction Level Modeling (TLM) [60,72] refers to modeling of a system at an abstract level, where architecture IPs are modeled at a functional level and the system bus is captured as an abstract 'channel' rather than the pin-accurate bus architecture or communication protocol. That is, in a TLM model the main focus is to analyze data

transfer between communicating modules, rather than on the way how the transfer can be accomplished. Starting from a model described at a TLM level of abstraction, where communication is characterized through the use of channels, one possibility would be to extract a set of figures of merit that may support the designer in the analysis of those aspects of the design useful for selecting the target architecture, which in turn implies, the decision on the type and number of computation elements (CPUs, DSPs, ASICs, etc.), the choice of communication resources (buses, FIFOs, etc.) and the hardware/software partitions.

3.1.1 Interface Refinement and Synthesis

In [114] synthesis of a system level bus interface has been presented for a single bus based architecture. Where a system can be viewed as a set of processes that communicate with each other over abstract communication channels. After hardware/software partitioning, a set of processes and variables of a system specification are mapped onto modules (CPUs, ASICs, memories, etc.) and channels are mapped onto system buses. The set of tasks performed to implement communication between the modules in a system are collectively defined as interface synthesis. The method called ”bus generation algorithm” that determines the bus width required for implementing a group of communication channels while minimizing performance degradation of the system processes. Such an algorithm incorporates system level constraints such as data transfer rates of the individual channels and the number of pins available to implement the bus. The algorithm allows the designer to explore a trade-off between the bus width and the performance of the processes communicating over the bus.

When multiple on-chip communicating modules share a single communication bus, a protocol needs to be defined in order to avoid any conflicts among modules during the communication. However, a communication protocol greatly influence the overall system performance and may lead to the violation of design constraints if the designers underestimate the actual communication load. In [48,49] Daveau et al. propose a communication synthesis approach that deals with both protocol selection and interface synthesis based on the allocation/binding of communication units. A communication unit is an object that can execute one or several communication primitives with a specific protocol, and it includes a controller that determines the communication protocol. The complexity of a controller may range from a simple handshake to a complex layered protocol. This approach allows for a wide design space exploration through the subsequent automated selection of communication protocols. In this approach, a system is modeled as a set of processes communicating through abstract channels, which executes a communication scheme invoked through a procedure call mechanism. The abstract channels act like as high-level communication primitives that

are used by the processes to communicate. Access to a channel is controlled by a fixed set of primitives and relies on remote procedures calls. A process that is willing to communicate through a channel performs a remote procedure call to a communication primitive (*send*, *receive*) of that channel.

During interface synthesis an implementation for each of the communication unit is selected from the implementation library and the required interfaces for all the process using the communication units are generated. The library may contain several implementations of the same communication unit, e.g., an interface with different protocols, different buffer sizes, and bus widths. Each communication is realized by a specific implementation selected from the library with regard to data transfer rates, memory buffering capacity, and the number of control and data lines. The synthesis algorithm first builds a tree of all possible implementations. This decision tree enumerates for each abstract channel all the communication units from the library that are candidate for allocation. The nodes of the tree are the abstract channels and the edges represent communication units that may implement that abstract channel. The leaves of the tree correspond to empty nodes.

To date, a complex SoC consists of several heterogeneous on-chip processing modules such as CPUs, ASICs, FPGAs, DSPs, and its design from the scratch is absolutely impossible due to the time-to-market constraint. Its sheer complexity makes it impossible to design everything from scratch in a reasonable time frame. Recently, the techniques such as ”design for reuse” and ”reuse of design” have been gained momentum in practice to reduce costs and shorten the time-to-market. However, the IPs may not have the same specification or the same implementation, which causes a huge overhead in terms of time and may cause major hindrances for successful integration. This motivates the need for tools to bridge the gap between the heterogeneous functional specification and its heterogeneous implementation. In [33, 137] the design environments PTOLEMY and CoWare are presented to integrate IPs with heterogeneous functional specifications. The CoWare synthesizes communication interfaces between hardware and software assuming that hardware/software partitioning of a complex system has been done efficiently. It allows the designer to specify and simulate communication channels at various levels of abstraction. Furthermore, it can be used to perform actual communication synthesis. This methodology allows for functional verification of a system but is not suited for the fast analysis of system communication throughput as this would require every interface in the system to be represented at the highest level of detail which, in turn, would lead to very long simulation time. Simply representing each interface at a low level of detail would not remedy this as the low detailed communication specifications are more or less abstract (primitive ports/channels, message passing, shared memory, etc.) and not tied to the particular protocol that will be used in the final system. The CoWare data model supports three communication mechanisms. Communication always happens between two ab-

stract processes. If these abstract processes are part of the same actual process, they are called intraprocess communication. If they are part of different processes, they are called interprocess communication. Intra-process communication is done by making use of shared variables and signals that are declared within the context of the process. Inter-process communication with a primitive protocol is based on remote procedure calls (RPC). On a master port the RPC function can be used to initiate a remote process. The RPC function returns when the slave process has completed data transfer. Similarly, Hines and Borriello [76] present the Pia co-simulation tool, which allows a designer to specify multiple communication models for each interface in the system and to dynamically switch between them during simulation. This way, a designer can choose to model some interfaces at a low level of detail and others (that he might want to debug) at a high level of detail.

In [163] Vahid and Tauro propose an object-oriented communication library (OOCL) for hardware/software co-design. The OOCL provides C/C++/VHDL send/receive communication primitives for numerous common protocols and components, with pretested underlying implementations. A designer can choose an OOCL channel supporting the desired protocol, without the need to focus on underlying implementation details. The user then instantiates a communication channel object, initializes it, and then sends or receives messages over it; all access to low level ports, registers, and communication behaviors are hidden within the implementation of an object. Because OOCL is a library, existing languages such as C and VHDL need not to be modified, and no synthesis tools are required to generate the communication behaviors. However, this approach focuses on specification and implementation rather than analysis.

In [91] Knudsen and Madsen present an approach to integrate communication protocol selection with hardware/software co-design. Their method finds the best system architecture, including the choice of communication protocols, the processing of data to be communicated, and the partitioning of system functionality onto the architecture. They claim that communication protocol selection must be done prior to the partitioning of a system into hardware and software. For example, pieces of computation that communicate small amounts of data compared to the amount of time spent in the computation should be isolated to processors with slow interfaces, while pieces of computation that communicate intensively with other pieces should be mapped to processors in such a way that they are linked to those pieces with fast interfaces, if they cannot be mapped to the same processor and if performance is a major concern. This means that the best system is not necessarily found when the protocol mapping is fixed initially and partitioning is performed later. Likewise, determining the best protocol mapping/configuration after the processors have been chosen and partitioning has been performed, will not in general result in an optimal system, as partitioning has been performed without knowledge of communication throughputs between system components and is, therefore, probably not optimal.

One of the major difficulties in reusing IPs lies in the different communication protocols they use [137]. When processors with incompatible protocols have to be interfaced, protocol conversions are required. A good protocol selection is possible if all processors involved in the communication are known because the choice is influenced by the number of factors such as the required speed and robustness of the data transfer. Hence, there is an potential growing need to standardize interface-based-design. Three factors drive the need for standards within the design and EDA industry [99]. These are (1) common communication principles, (2) common design formats, and (3) a unified approach to design quality measurement and assurance. The VSIA group has proposed three emerging system-level-integration standards that are already gaining industrial adoption. These are the system level interface behavioral document (SLIF) standard, the on-chip bus virtual component interface (OCB VCI), and the system level data types standard. The first of these is a mutual comprehension standard for rigorous interface based description of any VC (virtual component). It enforces a system level view upon standard VC integration, and provides the link between abstract models and VC implementation. The second and third standards are interoperability standards, and both tie in with the use of the SLIF standard. The OCB VCI transaction-level view provides a bus-interface abstraction that is not limited by the VC. The standard data types permits quick analysis of interoperability requirements, and guarantees that a common interpretation of data operations is used within the VC behaviors.

3.1.2 Trace Transformation Techniques Based on Khan Processes

There has been already a significant amount of work done in the area of on-chip communication architecture exploration and synthesis based on Khan processes [89]. In the Khan model, concurrent processes communicate using unbounded FIFO channels. Each process performs sequential computation on its private state space. The computation actions of a process are interleaved with communication actions that read data from input channels and write data to output channels. The Khan model fits nicely with signal processing applications as it conveniently models stream processing and as it guarantees that no data is lost in communication. Khan process networks are deterministic, i.e., the data stream that travels along each channel is determined by the input data; it does not depend on the order in which the processes are executed. As a result, application programmers can easily combine processes into process networks. Dataflow process networks are a special case of Khan process networks. The Khan and dataflow process network models permit applications to be modeled relatively independent of a specific target architecture. This enables reuse of application models and permits companies to build libraries of reuable functional IPs. In particular, the primitives used for communication between processes abstract from implementation aspects that need to be addressed later in the design trajectory. There is no need to

worry about issues such as synchronization with other processes, physical locations of buffers, or sharing of interconnect or memory resources. In the Khan process model, the read operation can not be initiated until data is available, when there is data then it is copied from the FIFO to the private state space of a process. However, for write operations there is no blocking, so a process can write its data from private state space to the FIFOs.

In [131] Pimentel et al. propose an environment called Artemis (architecture and methods for embedded media systems) to explore and synthesize communication bus architectures. The main goal of their work is to develop an architecture modeling and simulation environment that provides methods, tools, and libraries for the efficient exploration of heterogeneous embedded systems architectures. The meaning of efficient in this context is that the environment enables rapid evaluation of different architecture mappings, and hardware/software partitioning at various levels of abstraction for a broad range of applications. Beyond that second, Artemis provides the possibility to explore the design space for the reconfigurable embedded computer architecture. Artemis uses the Khan process network computational model which is obtained by restructuring a sequential application written in C/C++ into a program that consists of parallel processes communicating with each other via unbounded FIFO channels. It uses trace driven co-simulation to analyze the performance of a system modeled at application level. Each process, when executed, produces a trace of events that represents the application workload imposed on the architecture by that particular process. Thus, the trace events refer to the computation and communication operations an application process performs. By executing the Khan model, each process records its actions to generate a trace of application events, which is necessary for driving an architecture model as shown in Fig. 3.1. An architecture model is based on components that represent processors or co-processors, memories, buffers, buses, etc. Simulation of an application model requires an explicit mapping of Khan processes and channels of the application model onto the components of the architecture model as shown in Fig. 3.1. In the figure it can be seen that a trace event queue routes the generated trace of application events from a specific Khan process toward a specific component inside the architecture model. The Khan process dispatches its application events to this queue, while the designated component in the architecture model consumes them. The designers can make design decisions like hardware/software partitioning, mapping of computation and communication onto the hardware components. The selection of communication protocol can be done using the Artemis, which further refines the architecture to find the best design solution. To facilitate the process of model refinement, the architecture model library should include models of common architecture components at several levels of abstraction. The simulation should refine application level model events to match the detail level present in the architecture model. However, to explore such a large design space at different levels of abstraction using simulation techniques is quite

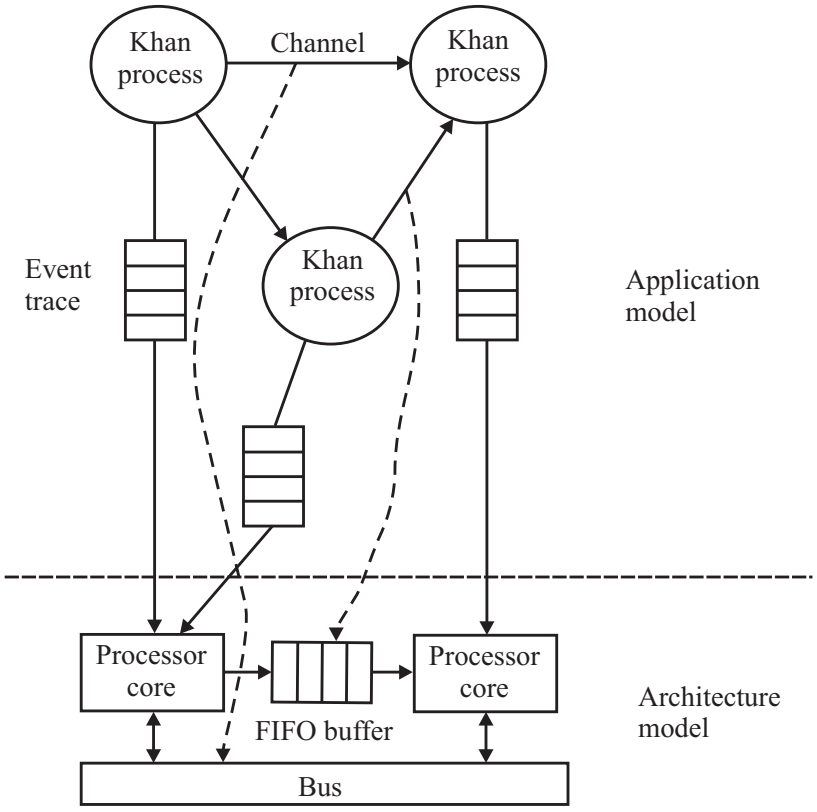


Fig. 3.1: Mapping a Khan application model onto an architecture model [131].

time consuming and increases design time-to-market constraint. Similarly, in [101] Lieverse et al. propose a technique called SPADE (system level performance analysis and design space exploration), for the architecture exploration of heterogeneous signal processing systems. The technique is very close to the Artemis [131], however, it focuses on the problems associated with the mapping of primitives used for expressing communication behavior at the application level onto primitives used to implement the communication architectures.

In the Artemis environment, mapping of an application model onto an architecture model is performed using trace driven co-simulation, where event traces generated by an application model drive the underlying architecture model. The abstract communication event of the application model, however, may not match the architecture level communication primitives, which, in turn, may lead to wrong design decisions. In [130] Pimentel et al. present a trace transformation method that is based on integer-controlled data flow (IDF) models [34], to perform communication refinement of application level events. They provide for mapping of the Khan processes from an application model onto architecture model components that supports the scheduling of application events from different event traces. Their proposition comprises an intermediate synchronization layer in between application and architecture level model for the scenario when multiple Khan processes are mapped onto a single architecture component. This layer consists of virtual processor components and FIFO buffers, which are for the communication inserted in between the virtual processors. The IDF model describes the internal behavior of a virtual processor. The incoming event traces from the application model specify *when* and *with whom* a virtual processor communicates, while the internal IDF model within a virtual processor specifies *how* the communication takes place.

Furthermore, in [31, 78, 69, 88] similar approaches as above are presented for the modeling of signal processing multimedia applications based on Khan process networks. One of the important issues in the interface synthesis is to determine the intermediate buffers in the interface required to temporarily store data to be transferred [92]. Kolks et al. address this problem by modeling a system as a set of processes similar to finite state machine. In [165, 166] on-chip communication traffic modeling and analysis for multimedia applications is presented. Their technique finds on-chip buffer space allocation and quantitative evaluations for a typical producer/consumer model.

3.1.3 Abstract Channel Model

Khan process network based on-chip communication analysis and synthesis has been used as a common technique to model applications at an abstract level with a set of

abstract communicating processes. However, it offers some limitations that is the use of infinite buffer size and the lack of write blocking operation in the buffer. Furthermore, abstract traces of an application model are used to map onto the components of the architecture level. This may not give the best mapping and can lead to bad design decisions. In [118] Nicolescu et al. propose an abstract channel model for the communication refinement. The concept of abstract channel is based on protocol fixed communication [49,72]. For communication refinement they use three abstract levels of communication, which are message level (protocol neutral communication level, ML), driver level (protocol fixed communication level, DL), and register transfer level (cycle accurate level, RTL). Their main contribution is to refine the communication from ML to DL. At ML modules communicate with each other by exchanging messages over ML channels with using generic data type messages. During channel refinement, the ML channel can be split into several DL channels. This process is called channel partitioning and it has to be done by the designer (probably who has to evaluate the trade-off between system performance and resource usage with using performance/cost models of channel implementations [32]).

In [45] Coppola et al. propose a design environment based on a C++ modeling library developed on top of SystemC [72], to support an object-oriented design methodologies, which separates IP modules into behavior and communication components and uses further two inter-module communication layers. The layering simplifies specification and allows further refinement by introducing application based abstraction. The bottom layer is called message box layer, which establishes inter-module transfer of interface signals and data according to generic or system specific protocols. The top layer is called communication driven layer and it translates inter-module transaction requests to the message box layer.

In [11] Abdi et al. present an automatic communication refinement engine for system level design. They assume that a system has been partitioned into hardware and software; and their behavior is modeled as an abstract communicating processes. The communication between the processes are modeled as an abstract data transfer and the tool transforms it to its actual bus level implementation. The main contribution of their work is to automate the transformation process. The inputs to the tool are an abstract communication model, a protocol of library including generic and processor specific protocols, and the synthesis decisions that guide the communication refinement engine. Inter-component communication is point-to-point and takes place through abstract channels, which support send and receive methods. The communication between components is modeled using three main schemes, which are two way blocking, one way blocking, and non-blocking. Similarly, in [14] Agosta et al. present static analysis of transaction level models. In this approach, a given system specification with an executable model is profiled to extract some computation metrics. These metrics are useful in the design space exploration phase, to define the main character-

istics of the hardware and software architecture.

The main task during communication analysis is the determination of all synchronization points so that all communicating processes hold the synchronicity condition. In [144] an approach is presented for the analysis of systems with parallel communicating processes for SoC design in order to determine the worst case execution timing behavior of a system.

In [74] Henkel and Ernst present Hw/Sw communication delay estimation for a shared memory architecture by separating a system into cluster of hardware and software. They estimate the communication delay by analyzing variables which are defined in a process and going to be used by another process. The approach is, therefore, more concerned to the Hw/Sw partitioning.

3.2 Bus Cycle Accurate Level Synthesis

The transaction level modeling approach provides functional validation of a system at an early phase of a complex system design flow, however, this model does not capture details about the on-chip communication behavior for the exploration of different design possibilities. Recently, some efforts [119, 172, 126] have been made with the concept of TLM that speedup simulation performance and apply them at bus cycle accurate (BCA) level. This model is used most commonly to capture IPs on a less detailed, functional level for improved simulation performance while modeling all the bus signals and timing accurately.

3.2.1 Real-time Constraint Driven Synthesis

The early works about on-chip communication bus synthesis and optimization are presented in [70, 100, 105]. Both approaches synthesize communication buses under real-time constraints. In [70] Grant et al. propose to synthesize communication buses for a simple signal processing algorithm, which consists of few adders, multipliers, and registers. In their synthesis algorithm, first, the operations are scheduled for given hardware resources to obtain a graph, where a node represents an operation and an edge between nodes represents the data dependency between them. A dependency between two nodes is equivalent to communication. All these communication activities among the operations are grouped together and mapped to communication buses. In addition to this, multiplexers are synthesized together with buses to share them among the communicating modules. Similarly, in [100] Li et al. present a technique to generate explicit communication from shared memory program references using a Crystal compiler approach. This approach starts off with a machine-independent high

level problem specification. A sequence of transformations, either suggested by the programmer or generated by the compiler, are then applied to this specification. These transformations are tuned for each particular machine architecture such that efficient target code with explicit communication can be generated. Their approach to compilation consists of three steps, which are control structure synthesis, data distribution, and communication synthesis. The first two steps are to generate appropriate communication from a machine independent system specification. The communication synthesis module consists of three main parts. The first part is analyzing reference patterns of the program and matching them with suitable communication routines. The second deals with scheduling and synchronization of send and receive pairs. The third part handles the partitioning onto the target processors and synthesizes communication buses, however, it is limited to a single shared memory architecture.

In a complex system design, designers frequently underestimate the peak load and synthesized communication bus architecture based on average communication requirements, which may lead to a bad design. Due to the peak load in a heterogeneous distributed embedded system, a custom communication topology is necessary to meet the real-time constraints. In [83,170] communication synthesis methods for distributed embedded systems are presented. In [170] Yen et al. propose a technique to synthesize a custom communication bus for arbitrary topologies in which point-to-point communication is a special case. Their synthesis algorithm selects the number of buses, the type of each bus, message transferred on each bus, and schedules the communication on the bus. A system is modeled using a task graph, which consists of set of processes with their dependencies. A process is a single thread of execution, characterized by computation time, which is a function of the module (processor, ASIC, etc.) to which it is mapped. A task is a partially ordered set of processes, which may be represented by as an acyclic directed graph known as a task graph, in which a directed edge represents a data dependency. For each task, a data transfer rate constraint, a hard deadline, a soft deadline, and a data size to be transferred are given as a problem specification. Furthermore, the bus is modeled assuming that each CPU has a local memory where the program code and local data are stored, such that local data and instruction fetching do not affect interprocess communication. When two or more processes are mapped onto different on-chip modules, the communication between them takes place over the communication bus, which introduces a delay in addition to the execution of the processes. Since the communication architecture is based on shared memory, sending process P_1 sends data to the shared memory and receiving process P_2 receives data from the shared memory. The duration to transfer data by a process includes the time spent on finishing an uninterrupted data transfer. It is proportional to the size of the message, the speed of the on-chip module, and the bus speed.

Fig. 3.2 depicts a task graph and communication processes after a system has been partitioned and mapped onto the appropriate modules of a SoC. The dashed boxes

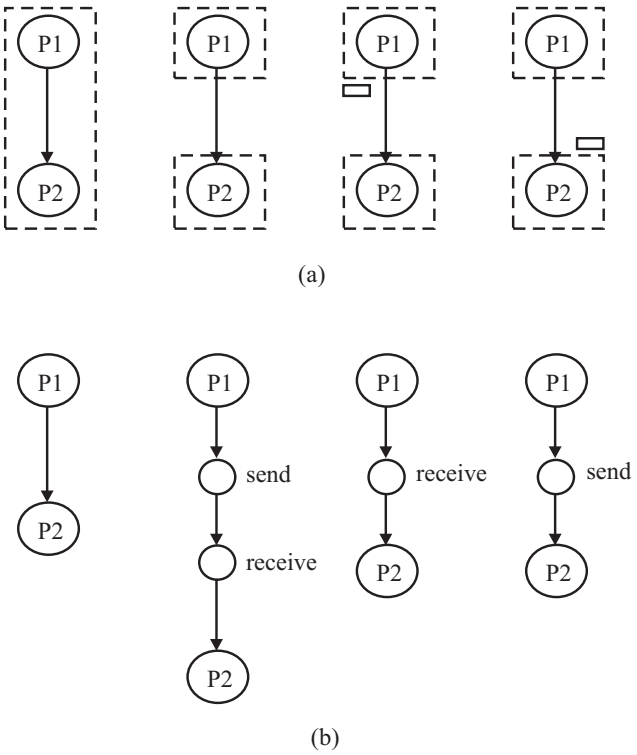


Fig. 3.2: Creation of communication processes for various communication schemes. (a) Task graph and allocation. (b) The corresponding communication processes [170].

represent on-chip modules and the small boxes represent dual-port buffers of on-chip modules. When two processes are mapped onto two different modules and connected by an edge, at least one communication process needs to be created for the corresponding message. If there is a dual-port buffer, either the sending or the receiving process can be deleted as shown in Fig. 3.2(a) and (b). The worst-case bus response time b_i, which is estimated as the longest time from the bus request of instance P_i to the instant P_i has finished all its data transfer. The total cumulative worst-case delay of communication processes is modeled as,

$$x = g(x) = c_i + \sum_{j=1}^{|P|} c_j \cdot \lceil x/p_j \rceil \tag{3.1}$$

where c_i is the computation time, c_j is the communication time for each communication process, and p_j is the period [97]. Their communication synthesis algorithm takes the estimated worst-case delay as a constraint and minimizes the total communication cost using the gradient-search method. At each iteration the algorithm maps each message (send or receive) to several communication buses and selects the one that does not improve the cost in presence of the worst-case delay constraint.

In [63,62] Gasteier et al. present an automatic generation of communication topologies at system level. The approach is limited to a single shared bus-based architec-

ture without arbitration scheme, which, however, requires deterministic data transfers among the processes in order to avoid bus access conflicts. They use VHDL processes and C programs to describe the behavior of a system at system level. These processes are communicating processes and they communicate with each other through abstract send and receive functions to exchange data. These abstract communications between processes are traced using co-simulation of the VHDL and the C model, which are used as input to the synthesis algorithm. For a given set of processes communicating via abstract send and receive functions and detailed information about the communication requirements of each process, the algorithm generates communication topologies using three steps: (1) clustering of transfers, (2) bus generation, and (3) final optimization.

At the first step of the algorithm, it merges all the traces of data transfer that do not overlap with each other into a single bus. This can be achieved by analyzing the data transfer time interval (start and end time) of all transfers, which do not conflict with each other in terms of timing. The worst case would be, when more than one process access the bus and transfer data at a same time. Let P^A be a set of processes that transfers data using a bus in cluster A and P^B be another set of processes that transfers data using another bus in cluster B then the clusters A and B can be merged into a single cluster *iff*,

$$\forall(P_i^A, P_j^B) \in P^A \times P^B \ : \ gcd(P_i^A, P_j^B) > 1 \tag{3.2}$$

where, $gcd(P_i^A, P_j^B)$ is the greatest common divisor. In the second bus generation step, the algorithm generates communication buses with the minimum cost for each cluster. They use the branch-and-bound method to find the minimum communication cost for tasks, which are scheduled for different bus widths. They use a recursive algorithm, which creates a search tree in depth first search order. The third step is the final optimization of the synthesized buses. In this step, the algorithm tries to merge buses if the total bus width of the merged bus is less than the sum of the bus widths of two buses. The algorithm is limited to find an optimal solution since it does not perform a complete search but the search is limited by two restrictions: first the imposition of a search order and second a quick look ahead communication cost estimation. This affects the selection of RAM, however, the search leads to an optimal solution with relatively high probability.

In [61] a communication synthesis approach is presented as an extension to [63,62]. This technique presents an iterative graph based on a clustering algorithm driven by a heterogeneous cost function, which takes into account bit widths, the probability of access collisions on the bus and the cost for arbitration logic. Similar to the previous approach, a system is specified as a set of communicating processes $P_1, P_1, \cdots, P_n$ and any two processes exchanging data with a certain frequency, which is called communication density $d_{i,j}$. The density $d_{i,j}$, $0 \leq d_{i,j} \leq 1$ is defined as the quotient of the number of clock cycles containing at least one transfer from process P_i to process P_j and the

number of clock cycles executed in total. Zero communication density $d_{i,j}$ means that no data is transferred between processes P_i and P_j, whereas a communication density of one means that data is transferred in one clock cycle. Furthermore, it is assumed here that the temporal distribution of transfers is not deterministic, i.e., the data transfer delay between two processes is not known in advance. The bus arbitration policy is implemented in the communication bus in order to avoid any simultaneous accesses by more than one process. The synthesis problem is formalized as an optimization problem that finds an initial solution, which consists of a set of clusters of communication processes. The costs for an initial clusters $\mathcal{C}$ is expressed as,

$$f_c(\mathcal{C}) = L \sum_{v_i \in V} P(v_i) + \sum_{C_i \in \mathcal{C}} \beta(C_i) + c_t \cdot max\{w(C_i)\}_{\forall C_i \in \mathcal{C}:|C_i|>1} + n_{arb} \cdot cost_{arb} \qquad (3.3)$$

In Eq. (3.3) the term L is a weight and it is set to a very high value in order to introduce high costs in case of port violations. The second and third terms are used for area and performance evaluation of the communication bus. The term $\beta(C_i)$ gives bus width needed for the logical bus assigned to cluster C_i and intuitively the bus width is a function of the area it uses, the total area due to the communication buses can be estimated by accumulating the bus width assigned to all clusters. n_{arb} is the number of arbitration units and its corresponding arbitration cost is $cost_{arb}$. The term c_t controls the trade-off between area and performance.

All above mentioned efforts synthesize either a single bus topology for all communicating modules or multiple communication buses with a single communication protocol. In a heterogeneous distributed embedded system all communicating processes may not need the same bandwidth and protocols, which results in under utilization or over utilization of communication resources, if a communication bus architecture is synthesized without a global consideration. In [123, 122] Ortega et al. present an approach that synthesizes communication buses for embedded systems with global consideration, which means the method analyzes bandwidth requirements for each communicating process from the given specification and clusters a group of communicating processes with bus width and communication protocol. This results in a hierarchical bus architecture with different communication protocols. More specifically, the method examines the problem of synthesizing communication for an arbitrary, yet static, bus topology. Instead of optimizing designers out of the design process, this approach allows designers to easily map their high-level designs to various implementation architectures for comparison. It helps a designer to explore quickly many more points in the design space than above mentioned techniques would allow. Their communication model is based on a set of processes that communicate by exchanging non-blocking messages. A non-blocking protocol is more suitable for distributed real-time systems than a blocking protocol partly because it decouples computation from communication [93]. A behavior description consists of a set of communicating processes. A process contains state information that may be used for intraprocess

communication. In addition to this description, the designer provides an architectural specification, which includes a list of processing elements, a mapping of the processes to the on-chip modules, a bus topology with bus protocols, and a mapping of messages to particular buses. The method analyzes the communication between processes and classifies it into interprocess communication and intraprocess communication. All interprocess communications are mapped onto the communication buses, however, intraprocess communications are mapped to a single on-chip module.

In [75] Hessel et al. propose approach to multi-level communication synthesis for heterogeneous multi-language systems based on a system modeling language SDL [18]. These methodologies allow to specify the intermodule communication at different levels of abstraction. In [150] Svantesson et al. propose a methodology and algorithms for efficient interprocess communication synthesis from a system level description in SDL. They implement SDL processes by two hardware blocks, namely a computation block and a communication block. The computation block implements the data computation function of the process as an extended FSM (finite state machine). The communication block implements the communication of a process with other processes. In this work, interfaces for interprocess communication are classified into five different types: send and forget (i.e., the source process just sends data and then continues to transfer the next), strobe based (the source process sends data along with a control signal and continues), handshaking based (the source process waits for acknowledgment from the destination process before continuing), FIFO based with a single instance of the destination process, and FIFO based with multiple instances of the destination process. Furthermore, each process is parameterized by process parameters, which include data size and timing related specifications. The algorithm analyzes the communication of each process P_i with all other processes that send data to it. If one of the transfers require a FIFO based communication interface then a communication module with a FIFO based module is selected from the library for that process.

In [65] Gogniat et al. present an extended communication synthesis method that provides characterization of communications and their implementation scheme in the target architecture. Their approach assumes that hardware/software partitioning and scheduling of a system have been done efficiently, which consists a basis for a back end of a co-design framework leading to hardware/software integration. The main aim of the work is to characterize the communications of an application in order to minimize resources and to determine the protocols. As this work focuses on the last step of a co-design flow for a dedicated static digital signal processing application, the underlying architecture consists of several heterogeneous cores such as e.g., DSPs, RISC processor, and memory. The behavior of a system is captured by a directed acyclic graph, where each node represents computation and an edge represents the data dependency between two communicating nodes. The dependency between two nodes is further classified as a temporal dependency and functional dependency. A temporal depen-

dency connects two nodes, which do not communicate over communication resources, and that are mapped onto the same on-chip module. In contrast to this, a functional dependency between two nodes means that they transfer data via communication resources. The communication synthesis algorithm starts tracing data transfer between all nodes with their functional dependencies and estimates the corresponding duration of the data transfer between sender and receiver as,

$$T_{com} = V_{data} \cdot \left\lceil \frac{L_{data}}{L_{bus}} \right\rceil \cdot N_c \cdot T_c \tag{3.4}$$

where, T_{com} is the data transfer duration, V_{data} is the number of data blocks to be transferred, L_{data} is the total size of data to be transferred, N_c is the number of clock cycles to access a data in the internal memory, T_c is the clock period of the communication interface, and L_{bus} corresponds to the internal memory bus width size. With the delay model given in Eq. (3.4) they compute an ASAP (as soon as possible) start time $t_{s_{ASAP}}$ and an ALAP (as late as possible) end time $t_{e_{ALAP}}$ of each node to calculate the mobility, which is defined as the difference between $t_{e_{ALAP}}$ and $t_{s_{ASAP}}$. If $t_{s_{ASAP}} > t_{e_{ALAP}}$ then the mobility is negative and the communication is asynchronous since there is no time overlap between the communicating nodes. Otherwise, the communication among them can be considered as synchronous.

In [160] Tsay et al. present the high-level synthesis of shared-bus systems from data flow graphs. They assume that hardware/software partitioning and mapping of a system onto the target architecture have been already performed. Based on the mapped system, computation and communication tasks are captured using co-simulation techniques. This results in a directed acyclic task graph, which consists of both computation and communication tasks. Graph nodes represent computation tasks and edges represent the communication between the computation tasks. The communication synthesis algorithm simultaneously performs scheduling, allocation, and binding of communication tasks to communication resources such as buses. The algorithm is implemented using three different approaches called the circular-arc coloring, integer linear programming (ILP), and channel assignment. Both circular-arc coloring and ILP approaches find the global optimal solution, however, their worst case run time complexity is **NP**-hard. In contrast to them, the channel assignment approach can be applied to perform scheduling, allocation and binding of communication tasks with run time complexity $O(|V|log|V| + r|V|)$, where *r* is a constant and called unfolding factor.

In [132] Pinto et al. propose a constraint driven communication synthesis method that enables automatic design of the communication architecture of complex systems using a library of pre-defined intellectual property (IP) components. The abstract model of a system consists of a set of computational models, which communicate through point-to-point unidirectional communication virtual channels. The algorithm

takes communication the constraint graph and a communication library as input and synthesizes communication buses. The communication constraint graph is captured from the abstract model of a system with a set IPs. The constraint driven communication synthesis problem is defined as a task to find a communication architecture that satisfies all the constraints specified as communication requirements on the channels. At the same time, this architecture minimizes a predefined cost function that captures an optimality criterion, which has to be defined for each specific application. A communication constraint graph is a directed graph, where each vertex is associated to a port of a computational system module and each directed arc represents a point-to-point communication channel between two modules. The arc is characterized by two parameters *d(a)* and *b(a)*, which are the arc length or distance and the communication bandwidth, respectively. Similarly, a communication library is a collection of communication links and communication nodes, where each node $n \in N$ has a cost $c(n)$ and while each edge is characterized by a set of link properties. The link length *d(l)* corresponds to the length of the longest communication channel that can be realized by the link. The link bandwidth *b(l)* corresponds to the bandwidth of the fastest communication channel and the link cost *c(l)* is defined with respect to the other links in the library based on an optimality criterion that varies with the application type. From a given communication constraint graph together with a communication library, an implementation graph is obtained. The cost of an implementation graph $\mathcal{G}^{'}$ is defined as,

$$C(\mathcal{G}^{'}) = \sum_{n^{'} \in N^{'}} c(n^{'}) + \sum_{a^{'} \in A^{'}} c(a^{'}) \tag{3.5}$$

where, $N^{'}$ is a set of communication nodes and $A^{'}$ is a set of arcs that correspond to the communication links between two nodes of the set $N^{'}$. In this approach there exists many possible graph implementations for a given library to satisfy the requirements given by a set of constraint graphs. It is guaranteed that there exists the optimum point-to-point implementation graph in any implementation graph and this graph is derived by implementing a single arc constraint independently from all the others present in the constraint graph.

In [138] Ryu et al. propose a custom communication bus generation for multiprocessor SoC designs. Their approach can synthesize bus architectures of five different types: (1) bidirectional first-in first-out bus architecture, (2) global bus architecture, (3) extended global bus architecture, (4) hybrid bus architecture, and (5) split bus architecture.

In a complex SoC, on-chip data traffic is not uniform over time. This is due to the diversity of applications to be run on a single embedded system. A communication bus architecture, which is synthesized for average data traffic conditions, is not able to handle peak traffic loads and may cause violations on the given real-time constraints. In [143] Sekar et al. describe FLEXBUS, a flexible, high performance on-chip communi-

cation bus architecture featuring a dynamically configurable topology. The FLEXBUS detects run-time variations in communication data traffic and efficiently adapts the topology of the communication architecture. It provides two different topology customization opportunities, first, dynamic bridge by-pass, which enables system level customization through run-time fusing and splitting of bus segments, and second, dynamic component re-mapping enabling component level customization through run-time switching of components from one bus segment to another. However, this configurable communication bus architecture offers several drawbacks in terms of an increase in logic and interconnects, configuration delay overhead, and maintaining compatibility problems with existing on-chip communication bus standards.

3.2.2 Layout and Floorplan Aware

In [156, 157] Thepayasuwan et al. present a layout conscious approach and bus architecture synthesis for the hardware/software co-design of SoCs optimized for speed. The method addresses layout related issues that affect system performance, such as the dependency between task communication speeds and interconnect parasitics. An embedded system is modeled as a quadruple consisting of a HDCG (hierarchical data and control dependency graph), resources, a floorplan, and a PM (performance model). The HDCG is an acyclic polar graph with one start node and one end node. It consists of three elements, which are a set of cluster nodes (CN), a set of communication cluster nodes (CCN), and a set of arcs. The cluster nodes represent tasks, functions, loops, and if-then-else constructs in the system specification. Each node in the cluster nodes is characterized by three parameters, which are start time, execution time, and end time. The CCNs represent data communication between CNs mapped to different processing units. The CCN is an alternating sequence of nodes corresponding to the transmission of data packets of a fixed size and nodes for synchronization. The resources are a set of IP cores available for the SoC implementation. The floorplan tree has a binary tree structure having the following two properties: first, leaf nodes correspond to IP cores and second, each internal node links the two nodes that exchange the maximum amount of data with each other. Lastly, the performance model symbolically describes the semantics of performance attributes, such as latency with respect to the invariant HDCG characteristics. The Performance model is a graph that contains three elements, which are first the starting node zero to set the modeled performance attributes to their initial value; second a constant part consists of linked symbolic variables and operational nodes, such as addition nodes, multiplication nodes, max nodes, and min nodes. The third and the last element of performance models is a variable part that includes additional directed arcs between the operational nodes.

The co-design methodology of this work includes three consecutive steps. The first

step partitions cluster nodes to processor nodes, binds operation nodes to functional unit cores, schedules cluster nodes, communication cluster nodes, and operation nodes and finds the speed requirements for communication cluster nodes. The second step decides about the IP core floorplanning, synthesizes the bus architecture, routes the buses, and characterizes the speed achievable on each bus. Finally the third step reschedules the cluster nodes, the communication cluster nodes, and the operation nodes without changing the partition or the bus architecture.

In [80] Hu et al. present a system level point-to-point communication synthesis using floorplan information. They assume that hardware/software partitioning and mapping onto the a set of IPs have been done and based on the profiling mapped system, communications among IPs are extracted to create communication task graph (CTG). In CTG, a node represents IP and an edge between any two IPs represents communication dependency, which is characterized by an amount of time to transfer data from one IP to another IP. The dependency between two IPs is classified as temporal dependency and function dependency. The temporal dependencies are the set of communication activities that take place within a IP, while the functional dependencies mean the communication activities occur between IPs. The communication-driven floorplanning algorithm takes a st of arbitrary shaped modules with their interconnection information and find a minimum area after placement with shortest wire-length. After the placement of modules, the communication synthesis algorithm finds the bus width for each point-to-point bus. The proposed algorithm is greedy algorithm which first calculate ASAP and ALAP start time of each communication with functional dependency and finds the lower bound of bus width to meet the real-time constraint between two IPs. Second step, algorithm checks whether the assigned bus width meets the given deadline or not. If it satisfies the condition then the algorithm terminates with an optimal bus width. Otherwise, it increases bus width and repeats the loop until the condition does not fulfill.

In [127, 128] Pasricha et al. present an automated synthesis methodology for on-chip communication buses with integrated floorplanning and a wire delay estimation engine to evaluate the feasibility of the synthesized bus architecture and detect timing violations early in the design flow. They assume that hardware/software partitioning and mapping of a complex system onto a set of IPs have been done efficiently. An embedded system is modeled as a graph called communication through put graph (CTG). This is a directed graph, where each vertex represents an IP and an edges connecting IPs represent communication between them. The floorplan engine is based on [12], which takes a list of components and their interconnections in the system and minimizes the total area associated with on-chip modules and their interconnection wires. The communication synthesis algorithm takes the CTG graph, a target communication architecture (e.g., AMBA [1]), a set of of communication parameter constraints, and a library of behavior IP models as inputs. The algorithm starts by some preprocessing

transformations on the CTG to that improve the performance of the entire system. Following this, it maps all the components from the CTG to a simple bus topology. Each node in CTG has information relating to the type of bus it can be connected to, which guides the communication bus mapping process. In the second step, the program iteratively selects a throughput constraint path (TCP) and searches the communication parameter space for a suitable parameter configuration and possibly performs topology mutations if needed until all TCP constraints are satisfied. When all TCP constraints are satisfied, the design is optimized in order to lower the communication cost and to make sure possible timing violations. In the next step the floorplanning and delay estimation engines are invoked in order to detect if there are any bus cycle time violations. If timing violations are detected, the algorithm is repeated once again, otherwise result is optimized in terms of timing and communication cost.

3.3 Post Synthesis Bus Optimization

At every abstraction level of a complex system design flow, analysis and optimization of communication behavior are important tasks in order to identify key design decisions for low power, better performances, and small in size etc. In the past, several research works have contributed to analyze and optimize communication architectures in terms of performance and power consumption for a given synthesized communication bus and its topology. This results look promising and are obvious addition for the post synthesis communication bus analysis and optimization.

3.3.1 Protocol Selection

In [133, 54] Pop, Eles et al. propose an approach for schedulability driven communication synthesis of time triggered embedded systems. They assume that an efficient hardware/software partitioning and mapping of computation and communication tasks onto the target on-chip module and communication buses, respectively, have already been performed. The approach is based on an abstract graph representation that captures at process level both data flow as well as the control flow. The time-triggered protocol [94] is used as the communication infrastructure for a distributed real-time system. They schedule the processes according to a static priority preemptive policy. They perform schedulability analysis on a given communication for four different types of messages, which are static single message allocation (SM), static multiple message allocation (MM), dynamic message allocation (DM), and dynamic packets allocation (DP). They then further show how communication protocol parameters can be optimized in order to fit the communication requirement of a given application to the given synthesized communication bus architecture. The optimization algorithm

performs both scheduling and parameter optimizations of a process and as a result, it generates an efficient bus access scheme for a communication bus.

For the schedulability analysis, firstly, they present a general approach for process scheduling with control and data dependencies considering a generic bus-based embedded system. The scheduling algorithm is based on list scheduling, which generates a schedule table for activation times, processes, and communications. This minimizes the worst case communication delay. Secondly, they investigate the impact of communication bus topologies and their protocols on the overall performance and demonstrate the importance of protocol selection to increase performance without any additional cost, by just optimizing the bus access.

In [90] Kim et al. propose an IP-based SoC synthesis framework with imprecise design costs for an SoC synthesis. The method is formulated in a probabilistic mixed integer linear programming (PMILP) model, which identifies design decisions such as selection of IPs, their assignment to communication buses, bus widths, and communication protocols. The PMILP formulation performs simultaneous IP selection, communication synthesis, and scheduling. The results show that the IP-centric design space with uncertainty can be explored successfully using the proposed framework.

In [96] Lahiri et al. present a method of design space exploration for optimizing on-chip communication architectures. Their approach optimizes a communication architecture by mapping a system onto a set of several available communication templates. These templates are standard on-chip communication bus architectures provided by vendors such as AMBA [1] and IBM CoreConnect [4]. Furthermore, they assume that the bus width and the topologies of communication bus architectures have already been determined and fed to their optimization algorithm. Based on those assumptions, they perform co-simulation of a given system for different bus templates and communication protocols and select the one that meets the real-time constraint.

3.3.2 Optimization for Low Power Consumption

There have been already a significant amount of efforts made in the area of system level approaches to reduce the energy of real-time distributed embedded systems. Dynamic voltage scaling and adaptive body biasing have proved to be an option to reduce energy consumption [85,162,37,57,82,67,58,21]. Since a heterogeneous real-time embedded system maybe realized in a single chip and may run a large diversity of applications, the workload offered to the system is not uniform over time. i.e., on-chip modules do not need to run at their highest speed for all times. When the workload offered to the system is low then the slack can be exploited by reducing the supply and body bias voltages. This results in a significant reduction of dynamic and leakage power consumption. There are lots of challenges and skepticisms concerning dynamic sup-

ply voltage scaling and body biasing techniques in terms of achieving efficient DC-DC voltage regulators and a dynamic workload detection unit. However, [68,81,47,35,73] present techniques to design DC-DC converters with 96% efficiency at the peak load of 134mW and dynamic workload detection units with a negligible area and power overhead. Overall, the voltage scaling technique can achieve a significant amount of power reduction for a system with variable workload over time.

In [19, 23, 77, 87, 38, 169, 39] dynamic voltage scaling and body biasing techniques are presented for processors and CPUs. Recently, in [16,17] Andrei et al. proposed a simultaneous communication and processor voltage scaling technique for dynamic and leakage energy reduction. They assume that a real-time distributed embedded system has been partitioned and mapped onto the appropriate modules of an SoC and that the on-chip communication bus architecture has been already synthesized. Based on the mapped and the target architecture, a directed acyclic task graph is extracted. The nodes of the graph represent the computational tasks, which compute data, while an edge between two computational tasks indicates the data dependency between them, i.e., the communication. They perform combined voltage scaling for both processor and communication buses with continuous and discrete voltage scaling schemes. The results show that combined supply voltage scaling and body biasing yield higher energy savings of around 30%.

In general, if the workload offered to a system is deterministic, the voltages are stored in a lookup table, which causes less delay overhead to exploit the slack. Since a real-time distributed embedded system runs several applications, the workload offered to a system is random in nature and this results in a stochastic behavior of dynamic slack. A major problem of the voltage scaling technique is to predict the amount of dynamic slack and exploit it with reduce on-line overhead. In [46] Cortes and et al. present a quasi-static assignment of voltages and optimal cycles for maximizing rewards in real-time systems with energy constraints. Their approach minimizes delay overhead subject to time and energy constraints.

In [79] Hsieh et al. propose an energy optimization of a communication bus architecture using a bus splitting technique. The parasitic resistance and capacitance are quite high in a long shared bus-based architecture. The timing and energy consumption of a long bus can be reduced by splitting it into segments. The bus splitting approach offers several advantages such as smaller parasitic load, larger timing slack, smaller driver size, lower energy consumption, and lower noise problems. They perform bus splitting after the on-chip modules have been physically placed on the bus and the bus wires have been routed according to their connections. The energy minimization problem for bus splitting is defined as a partitioning the on-chip modules into two equal sized sets such that the average energy consumption per clock cycle of the split bus architecture is at the minimum.

Normally the capacitances at the I/O pads need to be large enough to drive several loads connected to it and this further increases the total parasitic capacitances due to their interconnections, which affect power consumption and causes big delays for signal prorogation. In [149,149] Stan et al. present a bus-invert coding for data and address buses I/O. They assume that on-chip communication bus architecture has been synthesized and on top of this, a bus encoding technique can be implemented to reduce the dynamic power consumption. The proposed bus inverting technique is twofold: either invert the data values on the bus by setting a control signal *invert = 1* or no conversion of data values setting a control signal *invert = 0*. The method computes the hamming distance of the present data value and the next data value. If the hamming distance is larger than n/2, (where n is bus width) then set the *invert = 1* else set the control signal to 0 and leave the content of bus equal to the next data value. Similarly, at the receiver side the content of bus needs to be inverted according to the control signal. The results show that the peak power consumption can be reduced by 50% and average power consumption can be reduced by 25%.

In [20,24] Benini et al. present a synthesis algorithm for power efficient communication bus interfaces. They propose a general-purpose encoder-decoder architecture that can be used to reduce bus transition activity for generic data streams with completely unknown statistical properties. In [112,111] a delay model is proposed for both inductively and capacitively coupled lines. Based on this model, bus encoding is applied for a point-to-point interconnect to improve throughput.

3.4 Summary

In this chapter we discussed different existing techniques for on-chip communication modeling, synthesis, and optimization at different levels of abstraction, which are transaction level, bus cycle accurate level, and post synthesis level. Due to the ever increasing system complexity, it is practically impossible to model communication behavior at an RTL level, which takes lots of efforts in terms of time and revenue. Thus, the modeling of a system at an abstract level is essential for the broad exploration of a large design space. Recently, the TLM is being commonly used to model a system at an abstract level, where the IPs (intellectual properties) are modeled at a functional level and the system bus is captured as an abstract channel rather than a pin-accurate bus architecture. The main focus of TLM is to analyze data transfer between communicating modules rather than to decide how the transfer can be accomplished. In TLM, the modeling approaches are mainly based on the Khan process model [89] and the abstract channel model [118]. Khan process network based on-chip communication synthesis has been used as a common technique to model an application at an abstract level with a set of abstract communicating processes. However, it offers some

limitations such as the use of infinite buffer sizes and no write blocking operation for the buffer. Thus, to cope with the above problem Nicolescu et al. propose in [118] an abstract channel model for the communication refinement.

As TLM modeling does not capture system implementation details, bus cycle accurate level modeling techniques are used to explore the rest of the design space left by the TLM. The BCA model captures IPs with a less detailed, functional level for improved simulation performance while modeling all the bus signals and timing accurately. All efforts in BCA can be categorized into real-time constraint driven synthesis, layout, and floorplan aware synthesis. The first technique synthesizes communication buses without considering implementation issues such as CMOS technology, placement, and routing. Although the proposed techniques are promising in terms of the synthesis result, they may not be applicable if there are timing violations after placement and routing. The efforts made in [157,127] consider effects of layout and floorplan at higher levels of abstraction and check for timing violation at early design phases.

After the synthesis of communication buses, several techniques are used to optimize its power and delay characteristics. Among them voltage scaling and bus encoding are promising to reduce the power consumption of communication buses.

Chapter 4

On-Chip Communication Bus Synthesis and Optimization

Contents

The on-chip communication bus architecture is an interconnection network, which integrates several on-chip modules and provides a mechanism to exchange data between them. The recent trend in system complexity shows that there is a growing demand of communication traffic on the communication architecture. At the same time, trend in technology scaling indicates that wires are increasingly vulnerable to power and performance [13]. Thus taking into account these trends, designing a custom on-chip communication bus architecture is a challenging task. Traditional approaches are mainly based on the synthesis of a single shared bus based architecture [62, 61, 63],

which often fails to meet the performance requirements. Thus we are driven toward the synthesis of complex on-chip communication architectures that range from multiple hierarchical buses to entirely different network topologies. In the past, several efforts have been undertaken to synthesize advanced on-chip communication bus architectures [96, 127] based on available communication templates such as AMBA [1] bus and CoreConnect [4], however, the synthesized communication bus architecture may not always be optimal in terms of the optimal bus width and the number of buses. In [138] an automatic bus generation for a multiprocessor SoCs is proposed. Their approach generates buses for a given bus width considering real-time constraints. However, they do not find a trade-off between the bus width and the number of buses. Similarly, [132, 170] describes algorithms to synthesize communication bus topologes for point-to-point communication architectures.

The work described in this thesis is to introduce on-chip communication bus synthesis and optimization techniques for shared multi-bus based architectures. An assumption for synthesis is that a system has already been partitioned and mapped onto the appropriate modules of an SoC and the software part of the system specification is implemented in software that runs on a standard processor while the rest of the system specification is implemented in synthesized hardware. These hardware and software modules communicate with each other by exchanging data through shared buses. Furthermore, we assume that the amount of data to be transferred from one module to another module is fixed. The problem of on-chip communication bus topology synthesis is classified further into two main sub problems namely scheduling, allocation, and binding problems [109, 59]. As a scheduling problem, we schedule communication tasks for different bus widths. While as allocation-binding problems, we bind each communication task to the synthesized communication bus. The presented work makes the following contributions:

- we demonstrate that the proper bus width selection influences the number of buses (communication topology). For this we schedule communication tasks for different bus widths and select the one that gives the optimal bus width and the number of buses under given real-time constraints.
- we profile a hardware/software partitioned system and model the static communication behavior in terms of bus access and data transfer. The communication behavior of each communication task is characterized by three parameters: average number of data transfer, transition density, and spatial correlation. Based on these parameters, we estimate the communication cost of each communication task and refine the synthesized communication bus topology by moving or swapping the modules from one bus to another bus.

This chapter is organized as follows. At first **Sec.** 4.1 describes a model for a partitioned

and mapped architecture, where tasks that run within on-chip modules and between modules are characterized as data processing tasks and communication tasks, respectively. **Sec.** 4.2 formulates the on-chip communication bus synthesis as a scheduling problem and presents two different scheduling methods which are the globally optimal solution and a heuristic. The optimal solution method is formalized in linear programming, which finds the global solution with exponential run time complexity. While the heuristic is based on tabu search and finds a near-optimal solution in a polynomial run time complexity. **Sec.** 4.3 presents bus topology synthesis and optimization algorithm, which takes an optimized schedule of communication tasks in terms of bus width and number of buses. The algorithm synthesizes the number of buses and their interconnections to communication tasks. Further, the synthesized on-chip communication architecture is refined by moving or swapping on-chip modules from one bus to other bus. Finally, **Sec.** 4.4 gives a summary of this chapter. The results that are reported in this chapter have been already published in [188, 181, 180, 175]

4.1 Task and Architecture Models

We consider embedded systems which are realized as a MPSoC architecture. Such a system consists of several on-chip processing modules such as general-purpose processor, an application specific integrated circuit (ASIC), or a field-programmable gate array (FPGA). These on-chip modules communicate with each other by transferring data through communication buses such as shared buses or point-to-point connections. We assume that Hw/Sw partitioning and mapping of tasks onto the appropriate modules of an SoC have been done efficiently as shown in Fig. 4.1(a). Based on these mapped tasks, a directed acyclic extended graph $G_E(T, E)$ is obtained to extract the data processing tasks τ and the data communication tasks c of a given application. In the extended graph, a node $\tau \in T$ represents the data processing task, which is mapped onto the on-chip module, while edge $e \in E$ indicates a data dependency between the tasks (i.e. communication).

4.1.1 Data Processing Task

Data processing tasks $\tau \in T$ of a directed acyclic extended graph $G_E(T, E)$, are a set of tasks that are mapped onto synthesized hardware of an SoC. These tasks are for the computation of data, for example, a set of tasks for the fast fourier transformation (FFT) computation. The execution time of tasks $\tau \in T$ can be expressed as,

$$w_\tau = \sum_{i=1}^{|T|} NC_\tau \cdot T_d \tag{4.1}$$

$$T_d = \frac{K_6 \cdot L_d \cdot V_{dd}}{[(1 + K_1) \cdot V_{dd} + K_2 \cdot V_{bs} - V_{th}]^{\alpha}} \tag{4.2}$$

where, T_d is a delay for one clock cycle, NC_τ is the number of clock cycles needed to execute a data processing task τ, V_{dd}, V_{bs} and V_{th} are supply voltage, body bias voltage and threshold voltage, respectively. The term α is a technology dependent parameter with range $1.4 \leq \alpha \leq 2$ and K_1, K_2 and K_6 are the fitting parameters. Since the tasks τ are mapped onto the synthesized hardware, we assume that supply and body bias voltages of each task are known and provided to the communication synthesis algorithm.

4.1.2 Communication Task

On the one hand, all communications that take place among the data processing tasks τ, which are mapped onto different on-chip modules, are called communication tasks c as indicated by the square in Fig. 4.1(b). On the other hand, data processing tasks τ that are mapped to the same on-chip module are merged to a single node as shown in Fig. 4.1(b) and there does not exist an edge between two processing tasks. This indicates that the tasks τ_i and τ_j do not communicate using an on-chip communication bus. The notation c is a communication task, which takes a certain duration to transfer data from one module to another module by using an on-chip communication bus. This duration is called a communication lifetime interval (CLTI), which shows for how long a task c uses a communication bus. Furthermore, each communication task has its start time and deadline to finish the task. From the extended graph $G_E(T, E)$, a directed acyclic communication task graph $G_C(C, \Pi)$ is obtained with the start node *S* and deadline node *dl* to schedule the CLTIs of the communication tasks. In the communication task graph, a node $c \in C$ is a communication task, while an edge $\pi \in \Pi$ gives the dependency between the communication tasks.

Fig. 4.1(c) depicts the communication task graph with the ASAP scheduling of CLTIs for a 16-bit wide bus with a deadline of 14ms. An edge between two nodes c_i and c_j is weighted with w, which is the data processing time of a task τ_i. This gives an early start time constraint for a successor c_j to transfer data using a communication bus. The execution delay w of data processing task τ is calculated from Eq. (4.1). Fig. 4.1(d) depicts the ALAP scheduling of the CLTIs for a 16-bit wide bus with a deadline of 14ms. In Fig. 4.1(c) and (d), there is a difference in ASAP and ALAP time for tasks c_2, c_3, c_6, and c_7. This difference between the ALAP and ASAP time of a communication task is called slack. It measures how free we are to schedule the communication task c_i into different time slots so as to maximize the sharing of communication buses.

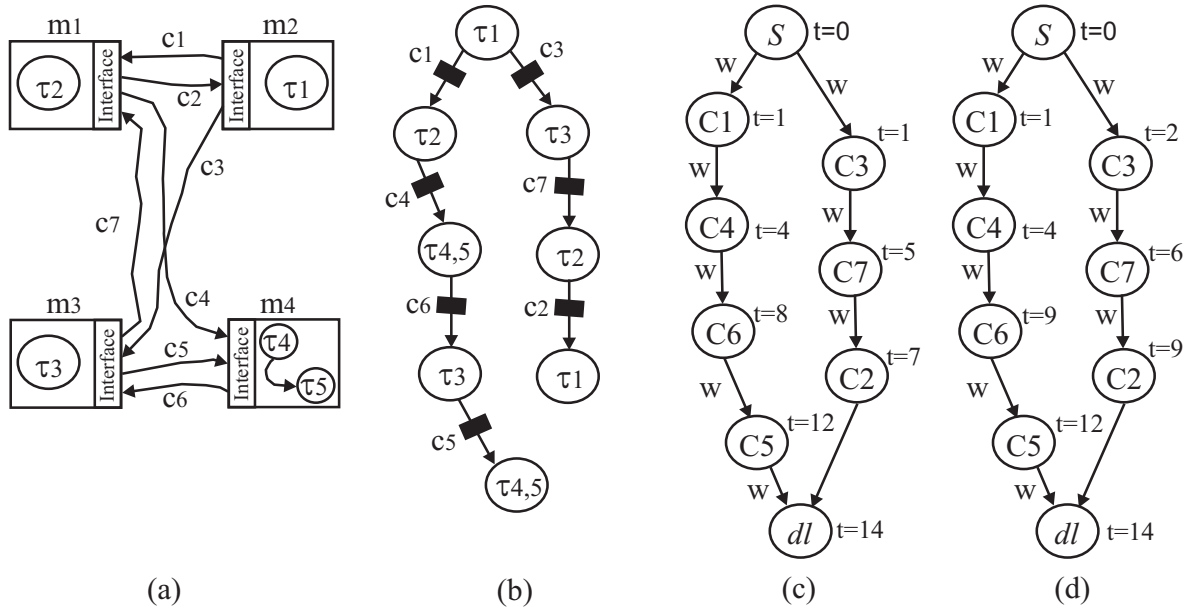


Fig. 4.1: Architecture model. (a) Target architecture with mapped tasks. (b) Extended tasks graph. (c) Communication task graph with ASAP scheduling of CLTIs for 16-bit wide bus. (d) Communication task graph with ALAP scheduling of CLTIs for 16-bit wide bus.

4.2 Communication Task Scheduling

In this section, we formalize the problem of on-chip communication bus synthesis as a scheduling problem, which takes a set of communication tasks and constraints as inputs and schedules them for different bus widths. As a result of scheduling, the method finds the optimal bus width. To find a trade-off between the quality of a solution and the run time complexity, we present two different scheduling algorithms: based on mathematical programming and tabu search.

4.2.1 Problem Definition

We assume that Hw/Sw partitioning of a complex system and mapping of their tasks to the appropriate module of an SoC have been performed efficiently. For each data processing task τ, its deadline dl, the number of clock cycles to execute the task NC_τ, supply voltage V_{dd}, and body bias voltage V_{bs} are given. From the target architecture with mapped data processing task(s) τ, a directed acyclic extended graph $G_E(T, E)$ is obtained by tracing the communication activities among the data processing tasks τ and results in a set of communication tasks c. Let C be a set of communication tasks and their data dependencies between the tasks are defined by a set $Depn \subseteq (C \times C)$, consisting of two-tuples (c_i, c_j) where a successor c_j depends on the results of the predecessor c_i. This data dependency between tasks is constrained by a set

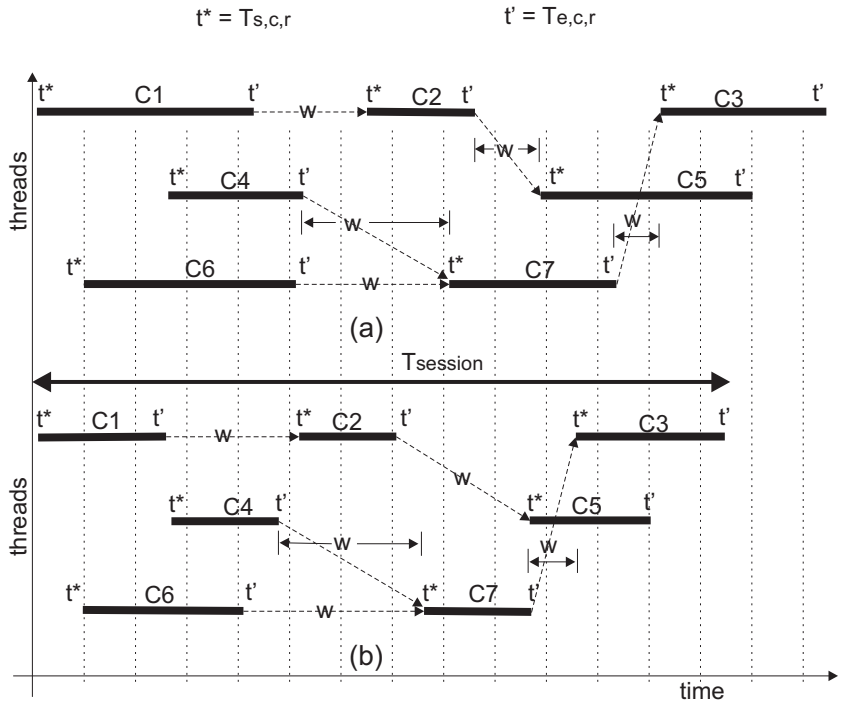


Fig. 4.2: Communication life time interval (CLTI) of on-chip modules. (a) Initial scheduling of communication tasks. (b) Optimized schedule of communication tasks in terms of the bus width and the number of buses.

$MinDelay \subseteq (C \times C \times W)$ consisting of 3-tuples (c_i, c_j, w) such that $\forall i, j \in [1 \ldots N]$, $(c_i, c_j)_{i \neq j} \in Depn | Depn \subseteq C \times C$, a task c_j can start transferring data no earlier than w time units after the completion of transferring data by c_i. Fig. 4.2(a) depicts an initial scheduling of communication tasks c for a *session* with bus width b_r and the data size to be transferred is NB_c (number of bit). The term *session* is a periodic time interval that includes all the possible traces of communication tasks and these traces will be repeated in all sessions. i.e., the amount of data to be transferred by each communication task c is fixed and all communication activities are deterministic. The dotted lines with arrows from one task to other task(s) in the figure show the data dependencies among them. The time interval between two communication tasks $(T_{s,c_j,r} - T_{e,c_i,r})$ gives an early start time constraint w for a successor c_j. The constraint w is an execution delay of mapped data processing tasks τ and the delay can be obtained using Eq. (4.1). The variables $T_{s,c,r}$ and $T_{e,c,r}$ are the start and end time of the communication task c with bus width b_r, respectively. These variables are evaluated for each $c \in C$ as a function of the bus width b_r, supply and body bias voltages, and data size to transferred NB_c. All communication tasks, which do not have predecessor, are defined as a set of tasks $c \in C$ called $StartTasks$ and their start time $T_{s,m_i,k}$ is unchanged and given as input to the model. In contrast to this, for the tasks $c \notin StartTasks$, the timing has to be re-evaluated because of their data dependency with the predecessor and successor tasks

whenever the bus width b_r changes.

Intuitively, the CLTI of a task c depends on the data size NB_c to be transferred and the bus width b_r. Since we assumed that the data size to be transferred by each task c is fixed, the CLTI of a task c is only a function of the bus width b_r. If communication tasks c are scheduled for different bus widths b_r, the number of overlaps among the CLTIs will not be the same. This is due to the fact that the ratio of change in CLTI's duration is different for communication tasks c with different data sizes NB_c. This accounts for the change in the number of overlaps among the CLTIs for different bus widths b_r. Each CLTI in Fig. 4.2(a) uses communication bus and our objective is to minimize the number of overlaps among the CLTIs such that all on-chip modules can transfer data using the minimum number of shared buses. Ideally, the number of overlaps can be made to zero for an infinite bus width, however, at an infinite bus width b_r, the utilization of the communication bus will be the lowest. Thus the communication tasks scheduling problem is an optimization problem. As an optimization problem, the communication tasks c are scheduled for different possible bus widths b_r in order to find the minimum number of overlaps and the minimum number of buses under the given constraint bus width b_r and real-time constraint. Fig. 4.2(b) shows the optimized CLTIs in terms of the number of overlaps, bus width, the number of buses, and the real-time constraints. We solve this problem of finding the optimal bus width and the minimum number of overlaps among the communication tasks using the mixed NLP formulation. According to graph theory, overlaps among communication tasks can be further classified into two different classes which are overlap and containment (OCT). Their definitions are as follows:

Definition 4.2.1 *An undirected graph is a pair G = (V,E), where V is a finite set, and E is a family of unordered pairs of elements of V. The elements of V are called the vertices of G, and the elements of E are called the edges of G.*

Definition 4.2.2 *A directed graph is a pair D = (V,A), where V is a finite set, and A is a finite family of ordered pairs of elements of V. The elements of V are called the vertices and the elements of E are called the edges of D. The vertices v and w are called the tail and the head of the edge (v,w), respectively.*

Definition 4.2.3 *An overlap graph is a pair* $G_o = (V, \mathcal{E}_o)$*, where a finite set* $V = \{v_i | v_i$ *represents an interval* $I_i\}$*, and a set* $\mathcal{E}_o = \{(v_i, v_j) | l_i < l_j < r_i < r_j\}$*. The values* l_i*,* l_j *and* r_i*,* r_j *are left and right points of the interval* i *and* j*, respectively.*

Definition 4.2.4 *A containment graph* $G_c = (V, \mathcal{E}_\kappa)$*, where a finite set of vertices* $V = \{v_i | v_i \; represents \; an \; interval \; I_i\}$ *and a set* $\mathcal{E}_\kappa = \{(v_i, v_j) | l_i < l_j, r_j < r_i\}$*. The values* l_i, l_j *and* r_i, r_j *are left and right points of the interval* i *and* j*, respectively.*

4.2.2 Optimal Solution

In this subsection, we present a scheduling model of communication tasks based on linear programming. The model takes a set of communication tasks and constraints as inputs and finds the global optimal solution in terms of number of OCTs among the communication tasks.

4.2.2.1 Minimizing OCTs Under Real-time constraints

Problem 4.2.2.1 *(Communication tasks scheduling (CTS) to minimize the number of OCTs among the tasks with hardware constraints) Perform a schedule of communication tasks $c \in C$ that minimizes $S = \sum_{o\in\mathcal{O}} \mathcal{N}_o + \sum_{\kappa\in\mathcal{C}} \mathcal{N}_\kappa$, where $\mathcal{N}_o$ and $\mathcal{N}_\kappa$ are the number of overlaps $o \in \mathcal{O}$ and containments $\kappa \in \mathcal{C}$ among the communication tasks, respectively; subject to: $\sum_{\forall c\in C}(t + CLTI_{c,r} + w) \leq T_{session}$ and $b_r^{LB} \leq b_r \leq b_r^{UB}$, for all time $t \in \{0, \cdots, \lambda\}$, where λ is the maximum possible time t to schedule communication tasks c in a session, $CLTI_{c,r}$ is the communication lifetime interval for a task c with a bus width r, w is the execution time of data processing tasks τ and the $T_{session}$ is the time constraint for a session. Furthermore, bus width b_r is constrained by its lower and upper bound b_r^{LB} and b_r^{UB}, respectively.*

We prove in Sec. 5.1.3 that Problem 4.2.2.1 is **NP**-hard. However, this problem can be solved in a *quasi-polynomial* time complexity for a few discrete values of b_r. The formulation of the communication tasks scheduling problem is given as follows:
Minimize:

$$\forall\, c \in C, \sum_{o\in\mathcal{O}} \mathcal{E}_o(c_i, c_j) + \sum_{\kappa\in\mathcal{C}} \mathcal{E}_\kappa(c_i, c_j) \tag{4.3}$$

where, $\mathcal{E}_o(c_i, c_j)$ and $\mathcal{E}_\kappa(c_i, c_j)$ are edges between two communication tasks c_i and c_j, respectively, depending on the condition as given below,

$$\mathcal{E}_o(c_i, c_j) = \begin{cases} 1 & if\ t^*_{c_i} < t^*_{c_j} < t^{'}_{c_i} < t^{'}_{c_j} \\ 0 & else \end{cases} \tag{4.4}$$

$$\mathcal{E}_\kappa(c_i, c_j) = \begin{cases} 1 & if\ t^*_{c_i} < t^*_{c_j},\ t^{'}_{c_j} < t^{'}_{c_i} \\ 0 & else \end{cases} \tag{4.5}$$

In Eqs. (4.4) and (4.5) $t^*_{c_i}$, $t^{'}_{c_i}$ are start time $T_{s,c_i,r}$ and end time $T_{e,c_i,r}$ for bus type r, respectively.

Subject to:

$$\forall c \in C, \sum_{\forall r\in R} X_{c,r} = 1 \tag{4.6}$$

The binary decision variable $X_{c,r} = \{0, 1\}$ indicates the scheduling of a communication task c; and is defined such that $\forall c \in C$ and bus width $r \in R$, $X_{c,r} = 1$ iff the real-time constraints are met for bus width b_r. i.e., Exactly one bus width b_r should be selected for each communication task c in order to meet the real-time constraint and the minimum number of OCTs among them. In this context, $r \in R$ is a library of on-chip communication buses, for example buses of 16, 20, 24,$\cdots$, 128-bit wide.

For each pair of communication tasks (c_i, c_j), where c_i is the predecessor and c_j is the successor in terms of data dependency. The start time $T_{s,c_j,r}$ to transfer data by c_j with the bus width b_r should not be earlier than w time units after the completion of data transfer by c_i.

$$\forall (c_i, c_j)_{i \neq j} \in Depn, \sum_{\forall r \in R} T_{s,c_j,r} \cdot X_{c_j,r} \geq \sum_{\forall r \in R} T_{e,c_i,r} \cdot X_{c_i,r} + w \tag{4.7}$$

In Eq. (4.7) start time $T_{s,c,r}$ and end time $T_{e,c,r}$ of each task can be calculated as,

$$T_{s,c,r} = \begin{cases} C(constant) & \forall c \in StartTasks \\ max(T_{e,c,r} + w) & \forall c \notin StartTasks \wedge \forall (c_i, c_j)_{i \neq j} \in Depn \end{cases} \tag{4.8}$$

$$T_{e,c,r} = \begin{cases} \left\lceil \frac{NB_c}{b_r} \right\rceil \cdot T_d & \forall c \in StartTasks \\ T_{s,c,r} + \left\lceil \frac{NB_c}{b_r} \right\rceil \cdot T_d & \forall c \notin StartTasks \wedge \forall (c_i, c_j)_{i \neq j} \in Depn \end{cases} \tag{4.9}$$

In Eq. (4.8), for all communication tasks $c \in StartTasks$, their start time $T_{s,c,r}$ is constant and given because the tasks $c \in StartTasks$ do not have any predecessor. However, the start time $T_{s,c,r}$ of tasks $c \notin StartTasks$ is not constant and their time has to be evaluated each time the bus width b_r is changed from b_{r_1} to b_{r_2}. In this case, the time $T_{s,c,r}$ is the maximum of the sum of $T_{e,c,r}$ for all c_i that are predecessors of c_j and their corresponding delay w due to data processing tasks τ. For example in Fig. 4.2(a) the start time of task c_7 is the maximum of $(T_{e,c_4,r}, T_{e,c_6,r})$ is $T_{e,c_4,r}$ and the maximum of $(w_{c_6,c_7}, w_{c_4,c_7})$ is w_{c_6,c_7}. Similarly, the end time $T_{e,c,r}$ of each task can be calculated using Eq. (4.9). For all communication tasks $c \in StartTasks$, $T_{e,c,r}$ is the ratio of the data size NB_c and the bus width b_r. For the tasks $c \notin StartTasks$, the end time $T_{e,c,r}$ of task c is the sum of start time $T_{s,c,r}$ and the delay to transfer data NB_c with bus width b_r. The term T_d is an α delay model of a CMOS transistor given in Eq. (4.2).

$$\begin{aligned} &\forall (c_i, c_j)_{i \neq j} \in Depn \wedge \forall c \in C \; and \; \forall r \in R, \\ &\sum_{c \in C} (t^* + CLTI_{c,r} + w) \cdot X_{c,r} \leq T_{session} \end{aligned} \tag{4.10}$$

The sum of the start time t^* of each communication task c, the data transfer times $CLTI_{c,r}$ of the communication tasks c with bus width r, and the data processing task execution delay w should be less than or equal to the given real-time constraint $T_{session}$

as shown in Eq. (4.10). Meanwhile the data transfer delay $CLTI_{c,r}$ is $(T_{e,c,r} - T_{s,c,r})$, which is also a function of the bus width b_r. The start time and end time of each CLTI are bounded by their constraints as given in Eq. (4.11). The start time $T_{s,c_i,r}$ of a module c_i should never be less than E_{c_i} and the end time $T_{e,c_i,r}$ of a module c_i should never be greater than L_{c_i}.

$$\begin{aligned} \forall c \in C, T_{s,c_i,r} &\not< E_{c_i} \\ T_{e,c_i,r} &\not> L_{c_i} \end{aligned} \tag{4.11}$$

In the above formulation, variables $\mathcal{N}_o$ and $\mathcal{N}_\kappa$ are integer variables, unlike the variables $T_{s,c_i,r}$ and $T_{e,c_i,r}$, which are not integer. The objective function is a summation of integer numbers, which is a linear function, while the CLTI is inverse function of bus width b_r, which is nonlinear with variable b_r, thus the overall scheduling and optimization problem is a mixed nonlinear optimization problem. This mixed NLP problem can be solved using any commercial convex optimization tool to find the global optimal solution.

4.2.2.2 Experimental Validation

We evaluate the effectiveness of the proposed techniques using an automatically generated benchmark, which consists of 64 communication tasks c and data to be transferred by the tasks ranges from 64 bit to 512 bit. The real-time constraint for a session $T_{session}$ is set to 370μs. Based on the mixed NLP (nonlinear programming) formulation proposed in subsection 4.2.2.1, we conducted an experiment to find the minimum number of OCTs among the tasks c with hardware constraints $16 \leq b_r \leq 64$ bit wide. The algorithm was implemented in C as a preprocessing model to interface with a convex optimization solver of MOSEK [5]. Furthermore, we consider a bus with 4mm in length and its corresponding single line capacitance for 70nm technology is 609fF [107]. Other technology dependent parameters for 70nm were adopted from [10], [2].

The results of Tab. 4.1 show that the number of overlaps $\mathcal{N}_o$ and containments $\mathcal{N}_\kappa$ among the communication tasks c change with bus width. In column 2 and 5 of the table, overall delay of communication tasks and total OCTs delay are presented, respectively. Furthermore in column 6, the amount of available slack of communication tasks also increases with increasing bus width. The minimum number of overlaps $\mathcal{N}_o$ and containments $\mathcal{N}_\kappa$ are found at bus width b_r = 64 bit wide, however, at this bus width, the bus will be underutilized with a total amount of available slack $\sim$43%. Hence, under the given real-time constraint of communication tasks $T_{session}$ = 370μs, the minimum number of overlaps $\mathcal{N}_o$ and containments $\mathcal{N}_\kappa$ are found to be 19 and 11, respectively with the bus width b_r = 40 bit wide. The amount of slack available at this bus width is 20.83%.

BusWidth (b_r)	($\sum t + CLTI_{c,r} + w$) (μs)	$\mathcal{N}_o$	$\mathcal{N}_\kappa$	$\sum \mathcal{D}_o + \mathcal{D}_\kappa$ (μs)	$\sum$Slack (%)	Run time (sec)
16	577	33	12	221	2.17	~ 14
20	507.8	26	13	160.4	5.43	~ 14
24	461.66	22	13	131	7.91	~ 14
28	428.71	24	12	111.57	11.31	~ 14
32	404	19	11	89	14.32	~ 14
36	384.77	21	10	72.11	17.18	~ 14
40	369.40	19	11	67.00	20.83	~ 14
44	356.81	20	9	53.45	23.53	~ 14
48	346.33	19	7	43.33	27.47	~ 14
52	337.46	19	7	38.61	34.19	~ 14
56	329.85	19	5	31.85	39.03	~ 14
60	323.46	20	5	27.73	41.92	~ 14
64	318.5	19	3	20.5	43.06	~ 14

Tab. 4.1: Number of overlaps among the modules for different bus widths

4.2.3 Heuristic Method

In this subsection we use tabu search as a heuristic method to schedule communication tasks. Tabu search learns from an adaptive memory and a reactive search process. The adaptive memory makes it possible to explore the solution area more efficiently by forbidding solution alternatives, which have been already visited for a certain time or with certain condition. This adaptive memory improves the efficiency of the exploration process, keeping track not only local information (such as the current value of the objective function) but also some information related to the exploration process. This systematic use of memory is an essential feature of tabu search (TS). The reactive search is a kind of feedback scheme that modifies the search parameters according to the search results is called reaction and is the core of the reactive search process. The focus of the reactive search method is on wide spectrum heuristic algorithms for discrete optimization, in which local search is complemented by feedback (reactive) schemes that use the past history of the search to increase its efficiency.

Let us consider an optimization problem with a given set S of feasible solutions and a function $f : S \rightarrow \mathbb{R}$, find some solution i^* in S such that $f(i^*)$ is acceptable with respect to some criterion (criteria). Generally a criterion of acceptability for a solution i^* would be to have $f(i^*) \leq f(i)$ for every i in S. In such a situation TS would be an exact minimization algorithm provided the exploration process would guarantee that after a finite number of steps such an i^* would be reached. In most contexts, however, no

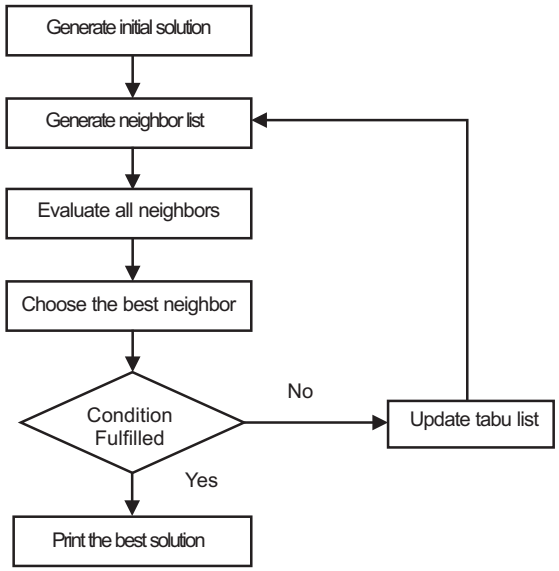


Fig. 4.3: A flow chart of tabu search heuristic

guarantee can be given that such an i^* will be obtained; therefore TS could simply be viewed as an extremely general heuristic procedure. TS begins like other local search methods with a valid initial solution and looks among all (allowed) neighbors for the best value of the objective function, even if this represents a worsening. The found neighbor solution is used as a starting point for the next iteration. Circling around a local optimum solution is avoided by setting already visited solutions to a tabu list. Because of the missing convergence characteristics of heuristics, it is necessary, as shown in Fig. 4.3, to stop this procedure with a subjective termination condition.

In this subsection, the communication tasks scheduling problem is formulated using tabu search heuristic. The first part of the formulation finds the minimum number of OCTs among communication tasks without considering the diversification. The second part of the formulation finds the minimum number of OCTs using diversification, which encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before.

4.2.3.1 Minimizing OCTs Under Real-time Constraint

In the previous subsection, we have discussed a linear programming based algorithm, which finds the global optimal solution of the CTS problem. As the computational time required for the algorithm grows exponentially with the size of the system, these kind of algorithms are impractical for big problems. Algorithm 4.1 gives a near-optimal solution of the CTS problem in a polynomial time complexity. The algorithm takes a directed acyclic extended task graph $G_c(C, \Pi)$, a real-time constraint for a session $T_{session}$, a *step* to find neighbors, a lower bound R^{LB}, and an upper bound R^{UB} as in-

puts. The heuristic starts by generating an initial solution. As mentioned in Sec. 4.2.1, the CLTI of an individual communication task c is an inverse function of the bus width b_r. Intuitively, for the largest bus width, the duration of CLTIs will be the shortest and this results in the minimum number of OCTs among the CLTIs. However, the bus utilization will decrease because of unused capacity of the bus. For the smallest bus width, however, the duration of CLTIs will be the longest, which results in the maximum number of OCTs among the CLTIs. In order to obtain a good solution, we generate an initial solution of the CTS problem randomly under the given lower and upper bounds of bus shown at line 8 of Algorithm 4.1. Line 10-34 of the algorithm is the main search loop and this loop repeats as long as termination condition is not fulfilled as shown at line 13 with a **while** condition. In this loop, first the neighborhood of b_r is determined, then the candidate list is generated and lastly the best candidate solution is selected from the candidate solution list and returns a directed acyclic extended graph $G_c^{min}(C, \Pi)$ with the minimum number of OCTs among communication tasks.

The definition of the neighborhood is an important task of a tabu search and it has to be defined in such a way that, the neighborhood can be determined by a slight modification of the present solution. Possible modification operators are swapping and temporal shifting of the CLTIs or use of different bus widths b_r such that the number of OCTs changes. In this case, the term shifting means moving the CLTIs to the right or to the left from its position in order to minimize the total number of OCTs. Because of the data dependencies among the communication tasks c, swapping of the CLTIs is not possible. Hence, the variation of the bus width b_r and the shifting of CLTIs are considered to generate the neighborhoods of a solution. Note that if the shifting of CLTIs is not possible, the use of the TS formulation is not suitable for the communication task scheduling problem. The neighborhood of a solution at b_r are the neighbor at b_r - *step* and b_r + *step*, including all the shifting possibilities as shown at line 18 and 22. The variable z is the possible number of neighbors; it depends on $|\mathcal{H}|$ as shown at line 16. At line 26, 27, 28 and 29, the total number of overlaps, the total number of containments, the overlap delay and the containment delay of individual overlaps and containments are evaluated, respectively for each neighbor $z.neighbor$.

In each iteration, tabu search finds possible neighborhoods and a set of candidates that are selected (and are put in the candidate list) from the neighborhoods to minimize the computation time. The CLTIs of each candidate list are examined completely in order to evaluate the number of OCTs among communication tasks. In general, there are two ways to select candidates from the neighborhoods. The first method is a random selection, which avoids repetition of the same neighbor and makes the tabu list short. The second is the deterministic method, which has a certain constraint to select candidates from the neighborhoods. In this CTS problem the deterministic method is used and a constraint to select candidates from the neighborhoods is the real-time constraint

```
FINDMINIMUMOVERLAPS(G_c(C, Π))
 1  n ← |C|;
 2  T_session ← GETDELAYCONSTRAINT();
 3  (R^LB, R^UB) ← GETBOUNDSOFBUS();
 4  H ← GETNUMOFNEIGHBOR();
 5  N_o(old) ← ∞;
 6  N_κ(old) ← ∞;
 7  step ← GETSTEP();
 8  InitialSolution ← GENRANDOM(R^LB, R^UB);
 9  b_r ← InitialSolution;
10  float shift1.neighbor[|C|] = ∅;
11  float shift2.neighbor[|C|] = ∅;
12  /*Beginning of tabu search heuristic*/
13  while (Condition ≠ true)
14  do
15      /*Determine neighborhood*/
16      for (c ∈ C) and (z = 1; z < |H|; z++)
17      do
18          if (z == 1)
19            then
20                 z.neighbor ← b_r − step;
21
22          if (z == 2)
23            then
24                 z.neighbor ← b_r + step;
25
26          z.N_o ← COMPUTENUMBEROFOVERLAP(z.neighbor);
27          z.N_κ ← COMPUTENUMBEROFCONTAINMENT(z.neighbor);
28          z.D_o ← COMPUTEOVERLAPDELAY(z.neighbor);
29          z.D_κ ← COMPUTECONTAINMENTDELAY(z.neighbor);
30          /*Determine candidate list*/
31          DETERMINECANDIDATELIST(G_c(C, Π), z.D_o, z.D_κ, z.N_o, z.N_κ);
32
33      /*Choose the best candidate solution*/
34      b_r ← CHOOSEBESTCANDIDATESOLUTION(CandidateList, T_session)
35
36  return G_c^min(C, Π);
```

Algorithm 4.1: Minimizing the number of overlaps.

of a session $T_{session}$. Line 31 of Algorithm 4.1 calls a function to select the candidate list. This function is shown in Algorithm 4.2, which takes a directed acyclic extended graph $G_c(C, \Pi)$, an overlap delay of an individual overlap $z.\mathcal{D}_o$, a containment delay of an individual containment $z.\mathcal{D}_\kappa$, the number of overlaps $z.\mathcal{N}_o$, and the number of containments $z.\mathcal{N}_\kappa$. The function returns a graph $G_c^{'}(C, \Pi)$ with a minimized number of OCTs among the CLTIs. Algorithm 4.2 performs a shifting operation for both over-

DETERMINECANDIDATELIST($G_c(C,\Pi), z.\mathcal{D}_o, z.\mathcal{D}_\kappa, z.\mathcal{N}_o, z.\mathcal{N}_\kappa$)
1 $G_c^{'}(C,\Pi) \leftarrow$ SHIFTOVERLAPTASKS($G_c(C,\Pi), z.\mathcal{D}_o$);
2 $\mathcal{G}_c^{'}(\mathcal{C},\Pi) \leftarrow$ SHIFTCONTAINMENTTASKS($G_c(C,\Pi), z.\mathcal{D}_\kappa$);
3 **return** $G_c^{'}(C,\Pi)$;

Algorithm 4.2: Determine the candidate list.

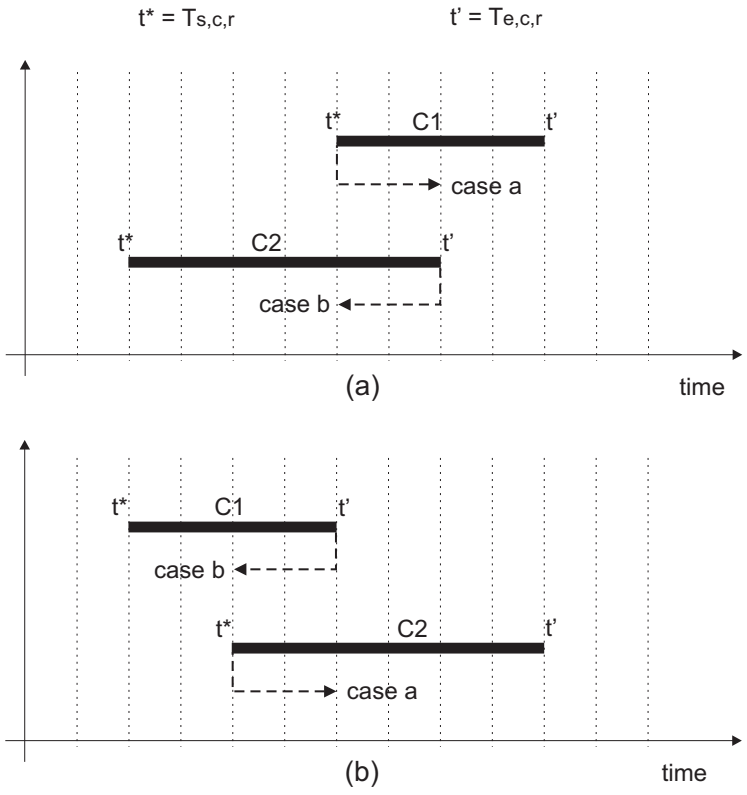


Fig. 4.4: Shifting possibilities for overlapped tasks.

lap and containment of the communication tasks as shown at line 1 and 2, respectively. Each of them returns an optimized graph in terms of their number of OCTs among the tasks $c \in C$.

Algorithm 4.3 performs the shifting of the CLTIs, which are overlapping with each other and minimizes the total number of overlaps among communication tasks. It takes a directed acyclic extended graph $G_c(C,\Pi)$ and a set of overlap delays of an individual overlaps $z.\mathcal{D}_o$ as inputs and returns a graph with the minimum number of OCTs. At line 10-11 of Algorithm 4.1, the candidate lists are declared with their corresponding shifted delay, which is an empty set $\emptyset$ at the beginning. A loop starts at line 1 and ends at line 16. In this loop, the algorithm checks for an overlap between communication tasks c_i and c_j as shown at line 3. If the condition is true then the minimum overlap delay d_o^{min} is selected from the set $z.\mathcal{D}_o$ and the corresponding pair of com-

```
SHIFTOVERLAPTASKS(G_c(C, Π), z.D_o)
 1  for (c_i ∈ C) and (c_j ∈ C)
 2  do
 3     if OVERLAP(c_i, c_j == true)
 4       then
 5            d_o^min ← GETMINOVERLAPDELAY(z.D_o);
 6            c_i ← G_o if (T_{s,c_i,r} < T_{s,c_j,r});
 7            c_j ← G_o if (T_{s,c_i,r} > T_{s,c_j,r});
 8            if SHIFTLEFT((c_i, d_o^min)== allowed)
 9              then
10                   SHIFTLEFT(c_i, d_o^min);
11                   shiftz.neighbor[c_i] ← −d_o^min;
12
13            else
14            SHIFTRIGHT(c_j, d_o^min);
15            shiftz.neighbor[c_j] ← +d_o^min;
16
17
18 UPDATEGRAPH(G_c(C, Π));
19 return G_c(C, Π);
```

Algorithm 4.3: Shifting of overlapped tasks.

munication tasks (c_i, c_j) are selected from the overlap graph $\mathcal{G}_o$, checking their overlap pattern as shown at line 6-7. The overlap pattern between communication tasks c is shown in Fig. 4.4. Fig. 4.4(a) depicts an overlap between two communication tasks c_1 and c_2, which have two different shifting possibilities. The first is called *case a*, which moves the task c_1 to the right and increases the start time $T_{s,c,r}$ and end time $T_{e,c,r}$ of all the successors tasks $c \in C$. The second shifting is called *case b*, which moves task c_2 to the left by exploiting the slack of task c_2. In both cases (a) and (b), the shift operation is performed without increasing the total number of OCTs. At line 8 of Algorithm 4.3, the possibility of moving a task to the left is checked and it is moved to the left only if a sufficient amount of slack is available for that move. If the condition is not satisfied then the task is moved to the right by default as shown at line 13-15 and the graph is updated with a new number of the OCTs at line 18. After completion of Algorithm 4.3, the shifting operation is performed for the containments, which is shown in Algorithm 4.4. It takes a directed acyclic extended graph $G_c(C, \Pi)$ and a set of containment delays for an individual containment $z.\mathcal{D}_\kappa$ as inputs and returns a graph with the minimum number of containments among communication tasks. The algorithm checks the condition for containment for each pair of communication tasks c_i and c_j at line 3, if the condition is fulfilled then it chooses the minimum containment delay d_κ^{min} from a set $z.\mathcal{D}_\kappa$ at line 5. A pair of communication tasks (c_i, c_j) are identified with the minimum containment delay d_κ^{min} from the containment graph $\mathcal{G}_{kappa}$ at line 6-7. From line 10-46,

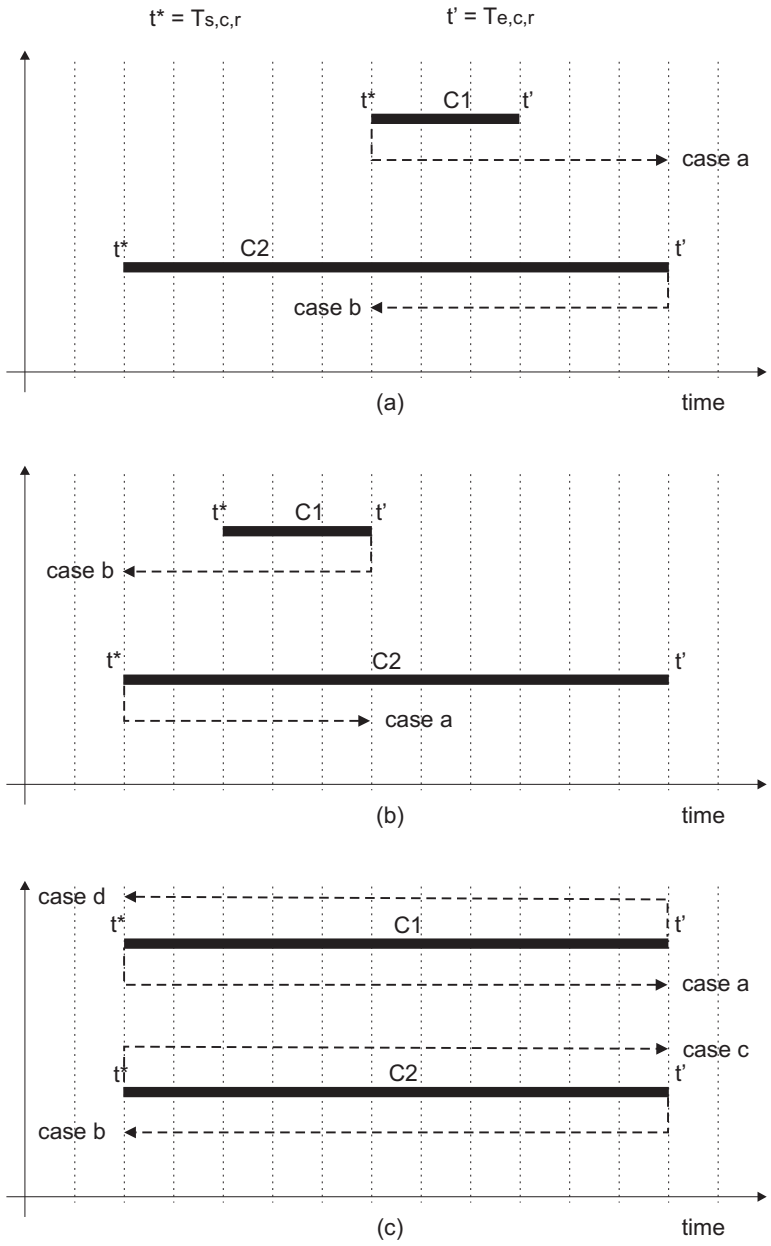


Fig. 4.5: Different pattern of containment and their shifting possibilities (a) Soft containment pattern type-I. (b) Soft containment pattern type-II. (c) Hard containment pattern.

the containment pattern is checked and the corresponding shifting of communication tasks c is performed. Fig. 4.5 depicts three different possible containment patterns between two communication tasks c and possible shifting operations. In Fig. 4.5(a), task c_1 has a containment with task c_2 and this containment between them can be avoided either moving task c_1 from left to the right (*case a*) or moving task c_2 from right to the left (*case b*). There are two more possibilities of shifting task c_1 from left to the right and task c_2 from right to the left in Fig. 4.5(a), however, the amount of delay for shifting is larger than for cases a and b. Hence, those possibilities of shifting are less probable to improve the containment than cases a and b, so we do not consider them to improve the computation time. The containment pattern shown in Fig. 4.5(a) is called soft containment pattern type-I and its condition is checked at line 10 of Algorithm 4.4. If shifting left of a task c_2 is possible, i.e., the amount of available slack of c_2 is sufficient for the shifting, then it is moved to the left without affecting the timing of its predecessor as shown at line 13-14. If the condition is not satisfied then by default task c_1 is moved from left to the right as shown at line 17-18. In this move (*case a*), the timing of the successors of task c_1 may or may not be affected depending on the amount of slack. Note that for shifting a task from right to the left, the overall delay remains constant, while shifting from left to the right, the overall delay of tasks may increase if the available slack does not compensate the increase in delay due to the moved task.

Fig. 4.5(b) depicts another soft containment pattern type-II of tasks c_1 and c_2 with a similar property as the pattern of Fig. 4.5(a). This pattern is checked at line 20 of Algorithm 4.4. At line 21, the condition of shifting a task to the left is checked and if the condition is satisfied the task is moved to the left, else the task is moved to the right by default as shown at line 27-28.

Fig. 4.5(c) shows another containment pattern of communication tasks c_1 and c_2 with an equal duration of CLTIs. There are four different shifting possibilities: *case (a), (b), (c),* and *(d)*. Intuitively, the possibilities of improvement in the number of containments by shifting either task c_1 or task c_2 to the extreme right end (*case a* and *c*) or to the extreme left end (*case b* and *d*), is lower in comparison to the soft containment pattern. This type of pattern is called the hard containment pattern, which is less probable for an improvement in the number of containments among the CLTIs. At line 31 and 36 of Algorithm 4.4, conditions are checked for shifting communication tasks from right to the left. If these conditions are not fulfilled then the default shifting (*case a* and *c*) will be performed as shown at line 42-45. After performing the shifting operation, graph $G_c(C, \Pi)$ is updated to get the new schedule of communication tasks $c \in C$.

After the completion of shifting overlaps and containments (mentioned in Algorithms 4.3 and 4.4 respectively), Algorithm 4.3 returns a graph $G_c^{'}(C, \Pi)$ with the minimum number of OCTs among communication tasks. At line 34 of Algorithm 4.1 the best candidate solution is chosen from the candidate list. The details of the algorithm

SHIFTCONTAINMENTTASKS($G_c(C,\Pi), z.\mathcal{D}_\kappa$)
1 **for** $(c_i \in C)$ **and** $(c_j \in C)$
2 **do**
3 **if** CONTAINMENT$(c_i, c_j == true)$
4 **then**
5 $d_\kappa^{min} \leftarrow$ GETMINCONTAINMENTDELAY$(z.\mathcal{D}_\kappa)$;
6 $c_i \leftarrow \mathcal{G}_\kappa$ **if** $(T_{s,c_i,r} > T_{s,c_j,r})$ **and/or** $(T_{e,c_i,r} < T_{e,c_j,r})$;
7 $c_j \leftarrow \mathcal{G}_\kappa$ **if** $(T_{s,c_i,r} > T_{s,c_j,r})$ **and/or** $(T_{e,c_i,r} > T_{e,c_j,r})$;
8
9 **switch** d_κ^{min}
10 **case** $(T_{s,c_i,r} - T_{s,c_j,r} > T_{e,c_j,r} - T_{e,c_i,r})$;
11 **if** SHIFTLEFT$((c_i, d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r}) == allowed)$
12 **then**
13 SHIFTLEFT$(c_i, d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r})$;
14 $shift\mathbf{z}.neighbor[c_j] \leftarrow -(d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r})$;
15
16 **else**
17 SHIFTRIGHT$(c_j, d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r})$;
18 $shift\mathbf{z}.neighbor[c_j] \leftarrow +(d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r})$;
19 **break** ;
20 **case** $(T_{s,c_i,r} - T_{s,c_j,r} < T_{e,c_j,r} - T_{e,c_i,r})$;
21 **if** SHIFTLEFT$((c_i, d_\kappa^{min} + T_{s,c_i,r} - T_{e,c_j,r}) == allowed)$
22 **then**
23 SHIFTLEFT$(c_i, d_\kappa^{min} + T_{s,c_i,r} - T_{s,c_j,r})$;
24 $shift\mathbf{z}.neighbor[c_j] \leftarrow -(d_\kappa^{min} + T_{s,c_i,r} - T_{s,c_j,r})$;
25
26 **else**
27 SHIFTRIGHT$(c_j, d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r})$;
28 $shift\mathbf{z}.neighbor[c_j] \leftarrow +(d_\kappa^{min} + T_{e,c_j,r} - T_{e,c_i,r})$;
29 **break** ;
30 **case** $(T_{s,c_i,r} == T_{s,c_j,r})$ **and** $(T_{e,c_i,r} == T_{e,c_j,r})$;
31 **if** SHIFTLEFT$((c_j, d_\kappa^{min}) == allowed)$
32 **then**
33 SHIFTLEFT(c_j, d_κ);
34 $shift\mathbf{z}.neighbor[c_j] \leftarrow -d_\kappa^{min}$;
35
36 **if** SHIFTLEFT$((c_i, d_\kappa^{min}) == allowed)$
37 **then**
38 SHIFTLEFT(c_i, d_κ);
39 $shift\mathbf{z}.neighbor[c_i] \leftarrow -d_\kappa^{min}$;
40
41 **else**
42 SHIFTRIGHT(c_i, d_κ^{min});
43 $shift\mathbf{z}.neighbor[c_i] \leftarrow +d_\kappa^{min}$;
44 SHIFTRIGHT(c_j, d_κ^{min});
45 $shift\mathbf{z}.neighbor[c_j] \leftarrow +d_\kappa^{min}$;
46 **break** ;
47
48 UPDATEGRAPH$(G_c(C,\Pi))$;
49 **return** $G_c(C,\Pi)$;

Algorithm 4.4: Shifting of the containment tasks.

```
CHOOSEBESTCANDIDATESOLUTION(CandidateList, T_session)
 1  /*Choose the best candidate solution*/
 2  for (r ∈ CandidateSolList)
 3  do
 4     bestSol = CHOOSEBESTCANDSOL(!Tabu);
 5     if (bestSol.T_session > T_session)
 6       then
 7            bestSol ← CHOOSEBESTCANDSOL(!Tabu);
 8
 9     if (bestSol.T_session < T_session)
10       then
11            solution ← bestSol;
12            Tabu ← solution;
13            N_o(old) ← GETNUMOFOVERLAP();
14            N_κ(old) ← GETNUMOFCONTAINMENT();
15
16
17  if (TabuList == Full)
18    then
19         DELETE(oldEntry);
20
21  return solution;
```

Algorithm 4.5: Choose the best candidate solution from the candidate list.

to select the best candidate solution are shown in Algorithm 4.5. The algorithm takes *CandidateList* and the real-time constraint of a session $T_{session}$ as inputs and returns the best solution (bus width b_r). At line 4, the algorithm chooses the best candidate solution from a set *CandidateList* with the condition that the chosen solution should not be tabu (already found solution). If the scheduling of communication tasks c with bus width *bestSol* gives an overall delay greater than the given real-time constraint of a session $T_{session}$ then the next best solution is chosen from the candidate list as shown at line 5-7. If the overall delay of a session is less than or equal to the given real-time constraint then the algorithm accepts *bestSol* as the best solution at line 11-12. The old number of overlaps $\mathcal{N}_o(old)$ and containment $\mathcal{N}_\kappa(old)$ are replaced by new numbers with bus width b_r = *bestSol*. In each iteration, TA finds the best solution and puts into the tabu list. At some point of the iteration, the list may get full due to the size limitation. At line 17, the tabu list is checked whether it is full or not. If it is full then the old entry is deleted from the list. Algorithm 4.5 returns the best solution at every iteration of tabu search as shown at line 21.

4.2.3.2 Extension for the Diversification Approach

So far we have discussed in the previous Sec. 4.2.3.1 that the Algorithm 4.1 finds a near-optimal solution of problem 4.2.2.1. But, the algorithm is inefficient in terms of the number of iterations required to find a near-optimal solution. In Fig. 4.4 and 4.5, we have seen the different overlap and containment patterns between the communication tasks c and their corresponding shifting possibilities. The shifting of a task c is performed to improve the number of OCTs. However, the shifting of a task does not guarantee the enhancement in the number of OCTs, if the overall delay of tasks (after shifting a task) violates the given real-time constraint of a session $T_{session}$. Among those overlap and containment patterns shown in Fig. 4.4 and 4.5, the hard containment pattern is more likely to violate the real-time constraint than other patterns. The Algorithms presented in above subsection finds the candidates with the minimum overlap or containment delay and performs the shifting operation at each iteration. If the overall delay of communication tasks after the shifting operation, is less than the real-time constraint $T_{session}$ then it accepts that as a candidate solution else it drops. This is done again and again for several iterations and it is more likely that the algorithm visits the same pattern of the same communication tasks, which has been already visited and performs the shifting operation to optimize the number of overlaps or containments between the tasks. Apparently, this makes the search algorithm inefficient to find a near-optimal solution. In this subsection, we present a diversification method to counter the problem of re-visiting candidates and improve the efficiency of the above proposed algorithm. The key idea behind diversification is to visit unvisited regions and to generate solutions that differ in various ways from those seen before.

```
CANDIDATELISTWITHDIVERSIFICATION()
 1  InitialSolution ← GENRANDOM(R^LB, R^UB);
 2  b_r ← InitialSolution;
 3  float shift1.neighbor[|C|] = ∅;
 4  float shift2.neighbor[|C|] = ∅;
 5  boolean forbid.task[|C|] = false ;
 6  /*Beginning of tabu search heuristic*/
 7  while (Condition ≠ true)
 8  do
 9     /*Determine neighborhood*/
10     for (c ∈ C) and (z = 1; z < |H|; z++)
11     do
12        if (z == 1)
13          then
14                z.neighbor ← b_r − step;
15
16        if (z == 2)
17          then
18                z.neighbor ← b_r + step;
19
20        z.N_o ← COMPUTENUMBEROFOVERLAP(z.neighbor);
21        z.N_κ ← COMPUTENUMBEROFCONTAINMENT(z.neighbor);
22        z.D_o ← COMPUTEOVERLAPDELAY(z.neighbor);
23        z.D_κ ← COMPUTECONTAINMENTDELAY(z.neighbor);
24        /*Determine candidate list*/
25        DETERMINECANDIDATELIST(G_c(C, Π), z.D_o, z.D_κ, z.N_o, z.N_κ);
26
27     /*Choose the best candidate solution*/
28     b_r ← CHOOSEBESTCANDIDATESOLUTION(CandidateList, T_session)
29
30 return G_c^min(C, Π);
```

Algorithm 4.6: Minimize the number of OCTs using diversification.

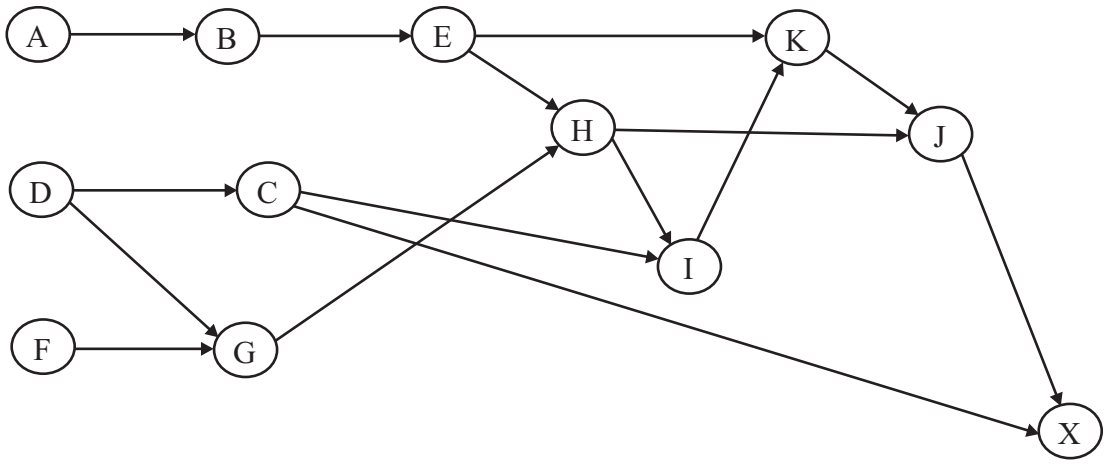


Fig. 4.6: Communication task graph

The proposed algorithm progressively conducts these steps: generate initial solution, generate neighbors, determine candidates, and choose the best solution similarly as the Algorithms presented in Sec. 4.2.3.1. However, at each iteration, if it finds that the overall delay of a shifted task with hard containment pattern violates the real-time constraint then the task is forbidden for the next iteration. This means, that in the next iteration the task is not considered for the shifting operation and the algorithm finds other unvisited neighbors for the shifting operation to minimize the number of OCTs among the tasks. Algorithm 4.6 is the proposed algorithm to determine the candidates with diversification. Line 1-4 are similar to the Algorithm 4.1, which declares the variables for a candidate list. At line 5 the algorithm declares a boolean variable and its contents for all communication tasks are set to *false*, which means at the beginning all tasks are allowed to visit in order to determine the candidates.

4.2.3.3 Evaluation of the Heuristic

In this subsection, we investigate the proposed heuristic for two different benchmarks and compare its results with the optimal solution, which was obtained using a mixed linear programming formulation. The benchmarks consist of directed acyclic extended graphs with 12 and 64 communication tasks c, which are generated randomly.

For the first part of investigation, we schedule the communication tasks shown in Fig. 4.6, using the heuristic with and without the diversification method. Tab. 4.2 depicts the results of the neighborhood search for benchmark-I with 12 communication tasks. The notations o and κ are overlap and containment types between two tasks, respectively. We chose b_r = 60 bit wide as an initial solution and its neighbors 56 and 64 bit were used to schedule the communication tasks at iteration 1. The algorithm selects the minimum overlap delay $\mathcal{D}_o$ and containment delay $\mathcal{D}_\kappa$ separately and checks their

	56 bit			64 bit		
Itr.	Task c	OCT Type	$\mathcal{D}_o$ *or* $\mathcal{D}_\kappa$ (μs)	Task c	OCT Type	$\mathcal{D}_o$ *or* $\mathcal{D}_\kappa$ (μs)
1.	A-D	κ	1.95	A-D	κ	1.83
	A-F	κ	1.95	A-F	κ	1.83
	B-D	***o***	**0.11**	B-F	κ	1.71
	B-F	κ	1.82	D-F	*o*	3.72
	D-F	*o*	4.03	**E-F**	***o***	**0.33**
	E-F	*o*	1.03			
2.	60 bit			68 bit		
	A-D	κ	1.89	A-D	κ	1.77
	A-F	κ	1.89	A-F	κ	1.77
	B-F	κ	1.76	B-F	κ	1.66
	D-F	*o*	4.82	**D-F**	***o***	**3.58**
	E-F	***o***	**0.32**			
3.	56 bit			64 bit		
	A-D	κ	1.95	A-D	κ	1.83
	A-F	κ	1.95	A-F	κ	1.83
	B-D	κ	**0.11**	B-F	κ	1.71
	B-F	κ	1.82	**D-F**	***o***	**3.19**
	D-F	*o*	4.06			
	E-F	*o*	0.34			
4.	52 bit			60 bit		
	A-D	κ	2.01	A-D	κ	1.83
	A-F	κ	2.01	A-F	κ	1.83
	B-D	κ	**0.27**	B-F	κ	1.71
	B-F	κ	1.88	**D-F**	***o***	**4.29**
	D-F	*o*	3.70			
	E-F	*o*	0.59			

Tab. 4.2: Neighborhood of benchmark-I without diversification

shifting possibilities. At each iteration the algorithm finds candidate solutions and puts them in the candidate list as shown in Tab. 4.3. In column 4, the minimum delay among the overlaps is 0.11 μs and among the containments is 4.03 μs. While in column 7, at bus width 64 bit, the minimum delay among the containments is 1.71 μs and among overlaps is 0.33 μs. Among the tasks, B and E were chosen as candidate solutions and put in the candidate list. In Tab. 4.3, there are two options, first, shifting task B from left to the right by 0.11 μs with bus width 56 bit, gives the total number of five OCTs, while shifting task E to the right by 0.33 μs with a bus width 64 bit, gives the total number of four OCTs. In this case, the algorithm chooses the one that meets the real-time constraints and produces the minimum number of OCTs, hence, the best solution at iteration 1, is b_r = 64 with four OCTs. At iteration 2-4, the algorithm repeats the same procedure and finds the best solution. In Tab. 4.3, it can be seen that the algorithm found task F as a potential candidate solution at iteration 2-4, however, task F was not chosen as a best solution at iteration 2 and 3, where it violated the given real-time constraint after shifting it to the right.

Tab. 4.4 shows the results of the heuristic, which was applied to the benchmark shown in Fig. 4.6 with the diversification approach. For iteration 1 and 2 the results are similar with the results shown in Tab. 4.3. After iteration 2, the algorithm sees that task F has a potential to improve the cost, however, it does not meet the real-time constraint when a shifting is performed to the right. So, task F is forbidden for the next iteration and the algorithm finds another candidate solution B for both bus widths 56 and 64 at iteration 3. Finally, at iteration 4, the algorithm finds the number of OCTs = 3 by shifting task E to the left with the delay -0.21 μs. The results, which carried out on the first benchmark conclude that the diversification method converges the search method faster and finds a near-optimal solution with few iterations. However, it is still a trade-off between memory and the quality of a solution.

The second part of the investigation was performed on a benchmark with 64 communication tasks, which is used in Sec. 4.2.2. The heuristic was applied to the benchmark with the diversification approach in order to compare the results of the optimal solution. Tab. 4.5 depicts the results of the heuristic for five iterations. In column 2, the best solutions (bus width b_r) are shown for each iteration. In column 3, the overall delay is shown for each iteration. In Column 4 and 5 the number of overlaps and containments are presented for each best solution. In column 6-7, the overlap and containment delays are presented for each iteration. As the objective is to find the minimum number of OCTs and the minimum bus width among the best solutions, we chose the 44 bit wide bus with corresponding number of 18 and 3 OCTs. The results seem to be promising compared to the results of Tab. 4.1 in terms of run-time, however, the bus width increased from 40 to 44 bit.

Neighbors					
Itr.	BusWidth	Task	Shifting in μs	$\mathcal{N}_o + \mathcal{N}_\kappa$	$\sum(t + CLTIc, r + w)$
1	56	B	+ 0.11	5	101.23
	64	**E**	**+ 0.33**	**4**	**101.60**
2	**60**	**E**	**+ 0.32**	**4**	**101.86**
	68	F	+ 3.58	3	105.43
3	**56**	**B**	**+ 0.11**	**5**	**101.23**
	64	F	+ 3.19	4	104.69
4	52	B	+ 0.27	5	102.39
	60	**F**	**+ 4.29**	**4**	**102.51**

Tab. 4.3: Candidate list of benchmark-I without diversification

Neighbors					
Itr.	BusWidth	Task	Shifting in μs	$\mathcal{N}_o + \mathcal{N}_\kappa$	$\sum(t + CLTIc, r + w)$
1	56	B	+ 0.11	5	101.23
	64	E	+ 0.33	4	101.60
2	60	E	+ 0.32	5	100.86
	68	F	+ 3.58	4	100.52
3	56	B	+ 0.11	5	102.00
	64	B	+ 3.19	4	100.86
4	52	C	+ 0.6	4	101.74
	52	E	- 0.6	4	101.69
	60	C	+ 0.21	3	101.26
	60	E	- 0.21	3	101.09

Tab. 4.4: Candidate solution with diversification

Itr.	BusWidth (b_r)	($\sum_{\forall c \in C} t + CLTI_{c,r} + w$) ($\mu$s)	$\mathcal{N}_o$	$\mathcal{N}_\kappa$	$\sum \mathcal{D}_o$ (μs)	$\sum \mathcal{D}_\kappa$ (μs)	*Slack* (%)	Run time (sec)
1	60	368.11	13	3	14.37	5.63	38.5	$\sim$3
2	56	364.83	15	4	17.84	8.29	29.7	$\sim$3
3	52	369.26	17	3	19.37	11.33	26.3	$\sim$3
4	48	367.92	18	3	23.93	13.16	24.2	$\sim$3
5	44	369.19	18	3	24.29	15.24	21.9	$\sim$3

Tab. 4.5: Number of overlaps among the modules with tabu search heuristic

4.3 Bus Topology Synthesis and Optimization Algorithm

The communication bus topology synthesis problem is a resource allocation and binding problems, which synthesizes the number of buses and the interconnections of on-chip modules to buses. If the number of synthesized buses are more than one then bridges are used to connect the buses. Since we use a heuristic to solve the allocation and binding problems, the synthesized buses are not always optimal. Thus we further optimize the communication bus architecture in terms of intra-module communication. i.e., the goal is to minimize communication between on-chip modules through bridges, since bridges are vulnerable to power consumption and delay overhead.

4.3.1 Topology Synthesis

After scheduling communication tasks $c \in C$ using either the optimal solution algorithm or a heuristic, the CLTIs with the minimum number of OCTs are obtained for a session with a bus width r. A set of communication tasks with the minimum number of OCTs are applied to the well known problem of graph partitioning called clique partitioning algorithm [161] to synthesize the communication bus topology. The topology synthesis means that the algorithm finds the number of shared buses and interconnections between buses and on-chip modules. Let $G = (V, E)$ denote a graph, where V is the set of vertices and E the set of edges. Each edge $e_{i,j} \in E$ links two different vertices v_i and $v_j \in V$. A subgraph SG of G is defined as (SV, SE), where $SV \subseteq V$ and $SE = \{e_{i,j} | e_{i,j} \in E, v_i, v_j \in SV\}$. A graph is complete if and only if for every pair of its vertices there exists an edge linking them. A clique of G is a complete subgraph of G. The problem of partitioning a graph into a minimal number of cliques such that each node belongs to exactly one clique is called clique partitioning.

Algorithm 4.7 is a heuristic, which is based on the algorithm proposed in [161] to solve the clique-partitioning problem. A super graph $G^{'}(S, E^{'})$ is derived from the graph $G(V, E)$, which is obtained after scheduling communication tasks. In graph G each vertex $v_i \in V$ represents an optimized CLTI of module m_i and there exists an edge $e_{i,j} \in E$ between two vertices v_i and v_j if and only if the CLTIs of two modules m_i and m_j do not overlap with each other. Each node $s_i \in S$ is a super-node that can contain a set of one or more vertices $v_i \in V$. $E^{'}$ is identical to E except that the edges in $E^{'}$ link to super-nodes in S. A super-node $s_i \in S$ is a common node of the two super-nodes s_j and $s_k \in S$ if there exist edges $e_{i,j}$ and $e_{i,k} \in E^{'}$. The function $COMMON\_NODE(G^{'}, s_i, s_j)$ returns the set of super-nodes that are common nodes of s_i and s_j in $G^{'}$. The procedure $EDGE\_REMOVE(E^{'}, s_i)$ removes all the edges in $E^{'}$ that have s_i as their end super-node. Initially, each vertex $v_i \in V$ of G is moved to a separate super-node $s_i \in S$ of $G^{'}$ in steps 3-4. At each step, the algorithm finds the super-node of the graph, where

```
TOPOLOGYSYNTHESIS()
 1  /*Create a super graph  G'(S, E') * /
 2  S ← ∅;
 3  E' ← ∅;
 4  for v_i ∈ V
 5  do
 6      s_i ← {v_i};
 7      S ← S ∪ {s_i};
 8
 9  for eache_{i,j} ∈ E
10  do
11      E' ← E' ∪ {e_{i,j}};
12
13  while E' ≠ ∅
14  do
15      /*Find*/S_{Num1}, S_{Num2}/*having most common node*/
16      MostCommons ← −1;
17      for e'_{i,j} ∈ E'
18      do
19          c_{i,j} ← |COMMONNODE(G', s_i, s_j)|;
20          if c_{i,j} > MostCommons
21            then
22                  MostCommons ← c_{i,j};
23                  Num1 = i; Num2 = j;
24
25
26      CommonSet ← COMMONNODE(G', S_{num1}, S_{num2});
27      E' ← EDGEREMOVE(E', S_{num1});
28      E' ← EDGEREMOVE(E', S_{num2});
29      /*Merge*/S_{Num1}andS_{Num2}/*into*/S_{Num1Num2}
30      S_{Num1Num2} ← S_{Num1} ∪ S_{Num2};
31      S ← S − S_{Num1} − S_{Num2};
32      S ← S ∪ {S_{Num1Num2}};
33      /*Add edge from*/S_{Num1Num2}/*to super nodes*/
34      for s_i ∈ CommonSet
35      do
36          E' ← E' ∪ {e'_{i,Num1Num2}};
37
38  return
```

Algorithm 4.7: Clique partitioning algorithm.

each super node consists of all the nodes in connected nodes s_{Num1} and s_{Num2} with the maximum number of common nodes. These two super-nodes are then merged into a single super-node, $s_{Num1Num2}$, which consists of all the vertices in s_{Num1} and s_{Num2}. The variable *CommonSet* consists of all the common nodes of s_{Num1} and s_{Num2}. All edges originating from s_{Num1} or s_{Num2} in G' are deleted. New edges are added from $s_{Num1Num2}$ to all the super-nodes in *CommonSet*. Above steps are repeated until there are no edges left in the graph. As an end result of this algorithm, we obtain a set of super-nodes with no edge, where each super-node $s_i \in S$ forms a communicating bus, which can be shared by a set of modules m_i inside of it.

4.3.2 Topology Optimization

In this section, we describe how a final refinement of communication topology is done by swapping modules from one bus to another bus on the basis of their communication cost. Fig. 4.7(a) depicts a scheduled communication tasks with the minimum number of OCTs among the CLTIs. When these optimized communication tasks are given to the clique partitioning algorithm, the heuristic gives more than one possible communication bus topology as shown in Fig. 4.7(b) and (c). This is because the heuristic takes into account only the information of overlaps to partition modules into buses. To choose the best solution among all possible bus topologies, we refine a communication bus topology using an intermodule communication profile. The main goal of topology refinement is to increase what we call the *locality of communication* such that communication overhead delay and power consumption will be minimized rarely using the bridge between two buses [95]. In Fig. 4.7(b) and (c) modules m_3 and m_5 are common modules, which do not overlap with rest of other modules. The refinement of the communication topology can be done by swapping them such that communication overhead delay and power consumption are minimized using the bridge. The criteria for swapping modules between the buses is the communication cost, which is a function of the communication behavior of a module ($A_{comm},\ \delta\ and\ S$).

4.3.2.1 Intermodule Communication Profile

The intermodule communication profile of a communicating task is characterized by three parameters: average number of communications A_{comm}, the transition density of communication (δ), and the spatial correlation of communications (S). These parameters are obtained by profiling a partitioned hardware/software system at a system level without any knowledge of on-chip communication bus architectures.

Let $c_i \in C$ be a communication task and its communication behavior is a set $CB \subseteq (C \times T \times V)$ consisting of three tuples (c, τ_z, v), where $T = \{\tau_1 \ldots \tau_n\}$ and $V = \{0, 1\}$

with $z \in [0 \ldots n]$ such that the relation $R : T \rightarrow V$ is a mapping of communication behavior at time instant τ_z to either 'ones' or 'zeros'. If for example at time instant τ_z the value of v is 'zeros', a communication task c_i does not communicate; and if it is 'ones' a communication task communicates with another task. This means that the communication behavior of a task for time instants $(\tau_1 \ldots \tau_n)$ is a sequence of 'ones' and 'zeros'. These values are obtained by profiling a system at system level on the basis of function call to transfer the data. Let $CB_{c_i}(\tau_z)$ represents the communication behavior of a task c_i with a sequence of 'ones' and 'zeros' at different time instants τ_z, where $i \in [0 \ldots l]$ then their intermodule communication profile can be obtained as follows:

$$A_{comm}(c_i) = \frac{\sum_{z=0}^{n} CB_{c_i}(\tau_z)}{n} \tag{4.12}$$

$$\delta(c_i) = \frac{\sum_{z=0}^{n} CB_{c_i}(\tau_z) \oplus CB_{c_i}(\tau_{z+1})}{n+1} \tag{4.13}$$

$$S(c_i) = \frac{\sum_{z=0}^{n} CB_{c_i}(\tau_z) \cdot CB_{c_i}(\tau_{z+1})}{n+1} \tag{4.14}$$

Where A_{comm} is the ratio of the sum of 'ones' to n = $|T|$, which corresponds to the average number of times a module uses communication resources. The transition density δ is the XOR operation of communication behavior values 'zeros' and 'ones' at time instants τ_z and τ_{z+1}. This shows how frequent a module switches between the two states of a bus being accessed or not accessed. Similarly, spatial correlation (S) is the AND operation of communication behavior 'zeros' and 'ones' at time instants τ_z and τ_{z+1}. This extracts the cluster of 'ones' or 'zeros' to indicate how continuously a communication task uses a bus.

4.3.2.2 Communication Cost

The total communication cost due to communication behavior of a task is evaluated as,

$$Cost(c_i) = K \cdot P_{comm}(c_i) + C_\delta \cdot \delta(c_i) + C_s \cdot S(c_i) \tag{4.15}$$

where,

$$C_\delta = MaxNumOf_{req} \times \frac{BridgeOverheadDelay}{PerAccess} \tag{4.16}$$

$$C_s = \frac{1}{MaxSizeOfBurst} \times \frac{BridgeOverheadDelay}{PerAccess} \tag{4.17}$$

$$\frac{C_\delta}{C_s} = \frac{MaxNumOf_{req}}{MaxSizeOfBurst} \tag{4.18}$$

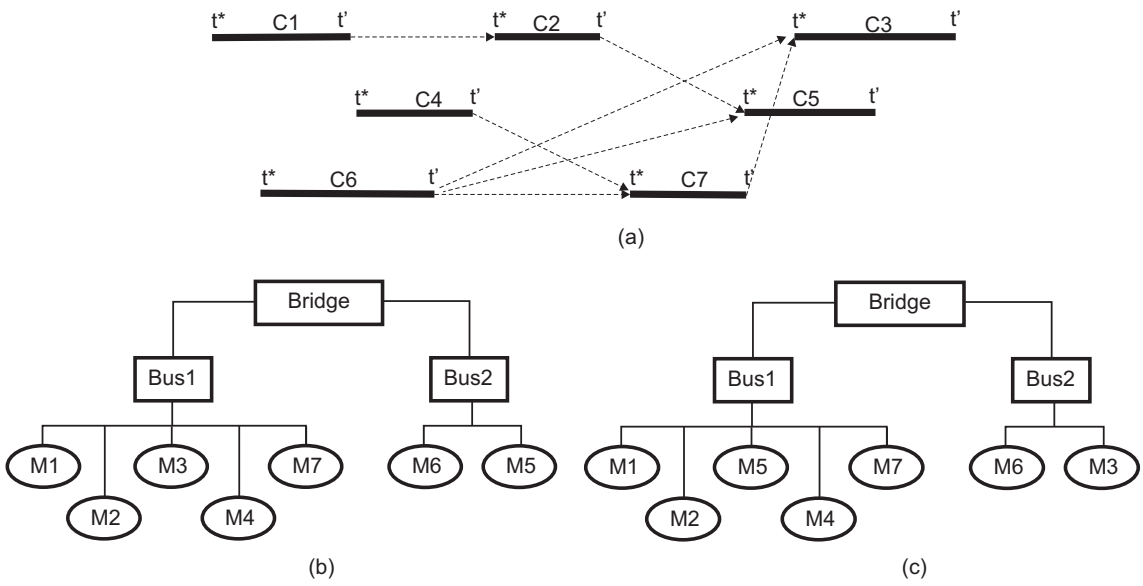


Fig. 4.7: The CLTI of modules and alternative architectures (a) an optimized CLTI of on-chip modules; (b) Synthesized communication topology (c) Alternative communication topology.

K = constant
C_δ = frequent bus access cost
C_s = bus uses cost for the contiguous transfer of data
$MaxNumOf_{req}$ = maximum number of bridge accessed in a session

The communication cost C_δ is a function of the number of times a bridge is accessed and the bridge overhead delay per access. The cost C_s is the inverse function of the max burst size. It is obvious that the cost $C_\delta > C_s$ because it is more expensive to transfer for a fixed data size by a communication task c_i with a high transition density and a low spatial correlation (contiguous transfer of data) than by a task c_j with a low transition density and a high spatial correlation of communication. Thus the total communication cost of c_i will be greater than the cost of c_j. This is because every time when there is a bus request, delay due to communication overhead includes delays such as bus request, bus grant delay, and the synchronization delay; and the parameters δ and S give the frequency of bus request and the contiguous transfer of data, respectively. From the given parameters of communication behavior and the costs, the total communication cost of each module is evaluated and compared with each possible module. If there is an improvement then we move or swap from one bus to another bus. For example in Fig. 4.7(b), if the total communication cost of a task in module m_5 is greater than the cost of task in m_3, we swap m_5 and m_3 so that the bridge will be rarely used and communication delay overhead and power consumption are effectively minimized.

4.4 Summary

When we analyze trends in system-on-chip design, on the one hand there is a big increase in system complexity, which puts high demands in terms of communication traffic on the communication architecture. On the other hand, device and wire scaling increases wire delays and power consumption and this is expected to worsen at each future technology node. As a result of these trends traditional single bus-based communication architectures [62] fail to meet the performance requirements. Until recently, several different types of on-chip communication architectures have been proposed ranging from multiple hierarchical bus based architectures to point-to-point interconnection architectures with different multiple communication protocols. However, communication bus architecture synthesis techniques presented in [96,127,128] do not find the optimal bus width, instead, they optimize the communication architecture by mapping a system into several available communication templates and choose the one that fulfills the requirements best. This can result in underutilization of communication resources. Further approaches presented in [132,170] focus on to the synthesis of optimal bus widths for any arbitrary point-to-point interconnection communication network. However, this is not the scope of our work.

In this chapter, the goal was to synthesize the optimal bus width and the number of buses for a shared multi-bus based architecture. An assumption for synthesis is that a system has been partitioned and mapped onto the appropriate modules of an SoC. Based on this assumption, the synthesis problem was formulated into three main subproblems, which are scheduling, allocation, and binding problems. As part of the scheduling problem, we presented two different methods, first one that gives a global optimal solution based on linear programming and second one that gives a near-optimal solution based on meta-heuristic algorithm called tabu search. To evaluate the effectiveness of the proposed synthesis techniques, we conducted experiments for both techniques using an automatically generated benchmark with 64 communication tasks c. The results show that the linear programming based formulation finds a global optimal solution in terms the number of OCTs among the communication tasks with a run time of approximately 14s. While the tabu search heuristic finds a near-optimal solution with run time of about 3s, however, the synthesized bus width is 44-bit wide, which is more than the solution of the linear programming formulation. In general the run time complexity of linear programming is exponential and it gets worse when the size of the problem is huge. Thus, the tabu search method can be applied to get a near-optimal solution in polynomial time complexity.

As part of the allocation-binding problem, we used well known clique partitioning algorithm, which takes a set of optimized communication tasks in terms of the number of OCTs and finds the number of buses and their interconnections with communica-

tion tasks. We further proposed a technique to refine the synthesized communication bus architecture using a static communication profile. The refinement is based on the principle of *locality of communication* so that the bridge is accessed rarely in order to reduce power and delay overhead due to the communication through bridge.

Chapter 5

Simultaneous Communication Bus Synthesis and Voltage Scaling

Contents

In **Chap.** 4, to cope with the ever increasing system complexity and the technology scaling, we presented a method to synthesize on-chip communication bus architectures with the optimal bus width and the number of buses without optimizing the energy consumption. Recently, the 2005 international technology roadmap for semiconductors (ITRS'05) [10] has reported that power and thermal aware design are the next big challenges for future technology nodes. As the feature sizes of devices and wires

shrink, power consumption per unit area increases. Its consequence is an increase in device temperature, which results in a reduction in carrier mobility and circuits speed. Thus design efforts for power optimization at each level of abstraction, in turn, rewards system performance and reliability [134].

There has been already a significant amount of work done in the area of system level approaches to reduce power consumption of real-time distributed embedded systems. Dynamic voltage scaling (DVS) and adaptive body biasing (ABB) can be options to reduce the energy consumption of a system as proposed in [162,37,57,35], where efforts were made to scale the voltage for only the processing units such as processors and CPUs, etc. As a result of this, dynamic power consumption decreases quadratically with the square of supply voltage and the leakage power decreases exponentially with scaled body bias voltage. Recently, DVS and ABB techniques were used to reduce the energy consumption of fat wires and repeaters based communication buses [16, 17], however, they assume the bus width, the number of buses, and the communication topology have been already synthesized and are given. Another possibility to reduce power consumption of a system is the usage of bus encoding techniques [148,149,20], which minimize the switching activities in a circuit and result a power efficient system. In general voltage scaling techniques can be applied only when the workload offer to the system is not uniform over time, meaning that the amount of slack can be exploited to scale the voltages. The bus encoding technique can be applied only when the transition density of data signals is high. However, due to the diversity of applications to be run on a single distributed embedded system, the workloads offered to it is rarely uniform over time. Thus, there is a lot of potential space for voltage scaling techniques to optimize the system's energy consumption.

In **Chap.** 4, we noticed that after scheduling communication tasks with the minimum number of OCTs, there is still a significant amount of slack left for the optimal bus width. This results in an underutilization of communication resources. Thus, in this chapter, we propose an extended model, which exploits the slack and performs simultaneous bus synthesis and voltage scaling in order to reduce energy consumption of communication buses. In comparison to above mentioned techniques, our main contribution is to integrate voltage scaling during the synthesis of on-chip communication buses and to find a trade-off between energy consumption and communication bus costs (bus width and the number of buses). The resulting synthesis problem is relaxed to the convex quadratic optimization problem and is solved efficiently using a convex optimization tool. The experimental results conducted on real-life examples demonstrate the synthesis of an energy efficient communication bus with total energy savings of up to 57.1% by scaling its supply and body bias voltages. Part of the results presented in this chapter have been published in [179,176].

The remainder of this chapter is organized as follows: **Sec.** 5.1 gives a brief ex-

planation about the target architecture model, motivation for simultaneous on-chip bus synthesis and voltage scaling, its problem formulation and the analysis of problem complexity. **Sec.** 5.2 presents a multi-bus based hierarchical communication bus model with voltage scalable driver and receiver. The bus architecture consists of islands of voltages and in between two islands an adapter is used to isolate them. **Sec.** 5.3 presents a mathematical formulation and optimization techniques for combined on-chip bus synthesis and supply voltage scaling. The formulation addresses both continuous and discrete voltage scaling of communication buses. Further **Sec.** 5.3 presents an extended model for body biasing so that both dynamic and leakage power consumption of communication buses can be reduced. Finally, **Sec.** 5.5 summarizes the work presented in this chapter.

5.1 Preliminaries

As in **Chap.** 4, we consider embedded systems which are realized as an MPSoC. Such a system consists of several on-chip processing modules such as general-purpose processors, ASICs or FPGAs. These on-chip modules communicate with each other by transferring data through communication resources such as shared buses or point-to-point connection. Further, we assume that Hw/Sw partitioning and mapping of tasks onto the appropriate modules of an SoC have been done efficiently as shown in Fig. 5.1(a). Based on these mapped tasks, a directed acyclic extended graph $G_E(T, E)$ is obtained to extract the data processing tasks τ and the data communication tasks c of a given application. In the extended graph, a node $\tau \in T$ represents the data processing task, which is mapped onto the on-chip module, while edge $e \in E$ indicates data dependency between the tasks (i.e. communication). All the communications that take place over the on-chip communication resource are captured by communication task c_i, as indicated by square in the Fig. 5.1(b). If the tasks τ_i and τ_j are mapped to the same module then there exists an edge between them without a square. This indicates that the tasks τ_i and τ_j do not communicate using an on-chip communication resource. The notation c_i is a communication task, which takes a certain duration to transfer data from one module to another by using an on-chip communication resource. Furthermore, each communication task has its start time and the deadline to finish the task.

From the extended graph $G_E(T, E)$, a directed acyclic communication task graph $G_C(C, \Pi)$ is obtained with the start node *S* and deadline node *dl* to schedule the CLTIs of the communication tasks. In the communication task graph, a node $c \in C$ is a communication task, while an edge $\pi \in \Pi$ gives the dependency between the communication tasks. Fig. 5.1(c) depicts the communication task graph with ASAP scheduling of CLTIs for a 16-bit wide bus with a deadline 7ms. An edge between two nodes c_i and c_j is weighted with w is the data processing time of a task τ_i, which gives an early

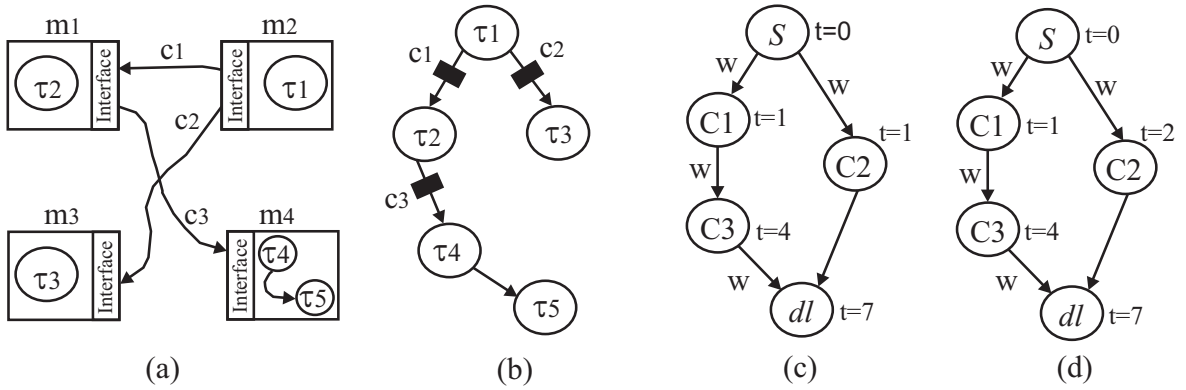


Fig. 5.1: Architecture model. (a) Target architecture with mapped tasks. (b) Extended tasks graph. (c) Communication task graph with ASAP scheduling of CLTIs for 16-bit wide bus. (d) Communication task graph with ALAP scheduling of CLTIs for 16-bit wide bus.

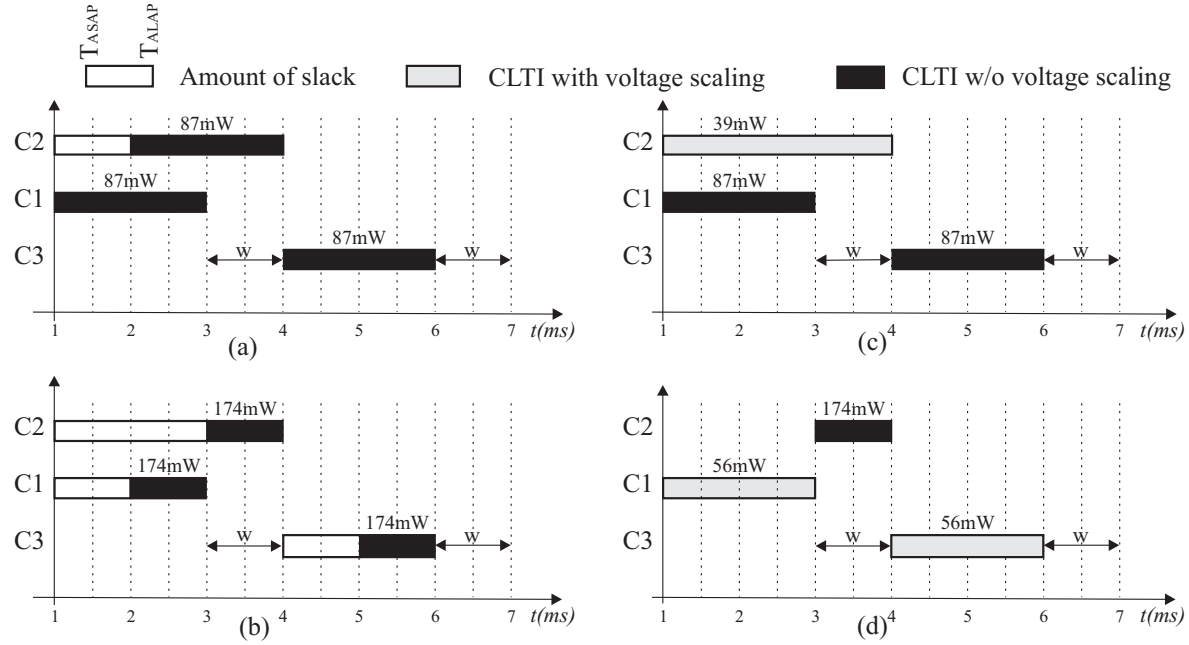


Fig. 5.2: Scheduling of CLTIs and voltage scaling of on-chip communication bus. (a) Scheduling of CLTIs for 16-bit wide bus. (b) Scheduling of CLTIs for 32-bit wide bus. (c) Scheduling and voltage scaling of CLTIs for 16-bit wide bus. (d) Scheduling and voltage scaling of CLTIs for 32-bit wide bus.

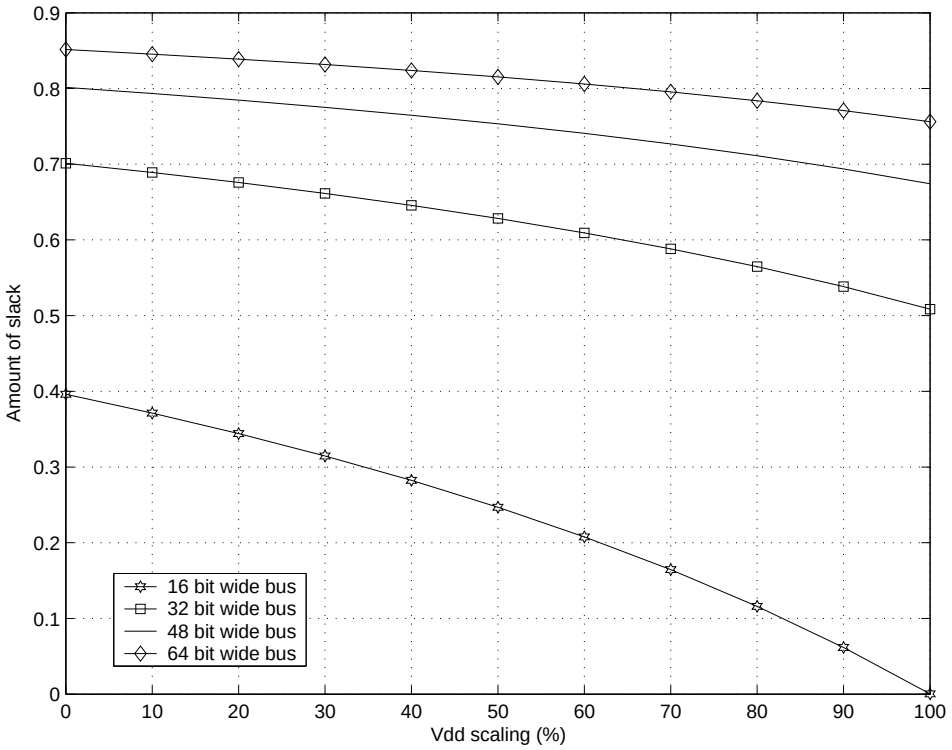


Fig. 5.3: Slack versus voltage scaling

start time constraint for a successor c_j to transfer data using the communication resource. Fig. 5.1(d) depicts the ALAP scheduling of the CLTIs for a 16-bit wide bus with a deadline 7ms. In the Fig. 5.1(c) and (d), there is a slack due to difference in ASAP and ALAP time for node c_2. This slack can be varied by scheduling communication tasks for different bus widths as,

$$slack_{c,r,V_{dd}} = t_{ALAP_{c,r,V_{dd}}} - t_{ASAP_{c,r,V_{dd}}} \tag{5.1}$$

where $t_{ALAP_{c,r,V_{dd}}}$ can be expressed as,

$$t_{ALAP_{c,r,V_{dd}}} = dl_c - CLTI_{c,t,r,V_{dd}} \tag{5.2}$$

$$CLTI_{c,t,r,V_{dd}} = \left\lceil \frac{NB_c}{b_r} \right\rceil \cdot T_d \tag{5.3}$$

In the above equations, dl_c is the deadline to finish a task, V_{dd} is the supply voltage and T_d is a gate delay for one clock cycle as shown in Eqs. (4.2). The dynamic energy consumption of each task c is given by [167, 107],

$$E_c = \alpha_\tau \cdot C_{eff} \cdot V_{dd}^2 \cdot T_d \tag{5.4}$$

where, α_τ is the switching activity of the communication tasks and C_{eff} is the effective switched capacitance for a data communication. The energy overhead, for switching

from V_{dd_i} to V_{dd_j}, is [167, 107]

$$\varepsilon_{i,j}^{\Delta V_{dd}} = C_r(V_{dd_i} - V_{dd_j})^2 \tag{5.5}$$

where, C_r is the capacitance of the power rail. The time overhead, for switching from V_{dd_i} to V_{dd_j}, is given by [107]

$$\delta_{i,j}^{\Delta V} = \rho|V_{dd_i} - V_{dd_j}| \tag{5.6}$$

where ρ is a constant. Fig. 5.3 depicts, the amount of slack of each communication task for different bus widths and voltage levels, considering a fixed data size. In this thesis, we perform the communication resource selection, the scheduling, and the voltage scaling for the communication task graph in order to synthesize the on-chip communication architecture.

5.1.1 Motivation

In order to motivate the principles behind the proposed techniques for the synthesis of on-chip communication architectures, we illustrate that the slack of communication tasks changes with the bus width and that slack can be exploited for the on-chip communication resource sharing and voltage scaling, which ultimately increase system efficiency in terms of energy consumption and chip size. Consider a system that has been partitioned and mapped onto the on-chip modules of an SoC and the driver of each module is capable to scale the supply voltage while transferring data from one module to another module. As shown in Fig. 5.1(a) first, module m_2 executes task τ_1 and its driver transfers data to m_1 and m_3 to execute tasks τ_2 and τ_3, respectively. After receiving the data from module m_2, module m_1 executes task τ_2 and its driver transfers data to module m_4, which executes tasks τ_4 and τ_5. The task τ_5 has to be finished before the deadline of 7ms. The ASAP and ALAP scheduling of the communication task graph in the above example with their start node and deadline node are shown in Fig. 5.1(c) and (d), respectively. Fig. 5.2(a) shows a scheduling of CLTIs with their ASAP and ALAP time of all the communication tasks c_1, c_2 and c_3, considering a 16-bit wide bus and nominal voltage settings (the highest supply voltage = 1.8V and body bias voltage = 0V), i.e., all drivers run at their maximum performance. This schedule of communication tasks for a 16-bit wide bus results in a slack (denoted by a white rectangle) of 1 ms and needs two separate buses to meet the time constraint of 7ms. From the given power consumption at the nominal voltage as shown in Fig. 5.2(a), the total energy consumption of all communication tasks can be calculated as 3·(87+87+87)mW·2ms = 522μJ. Fig. 5.2(b) shows the scheduling of the same communication tasks c_1, c_2 and c_3 for a 32-bit wide bus and the nominal voltage. This schedule results in the total slack of 4ms, which increases the mobility so that all communication tasks can share a single bus. The total energy consumption at the nominal voltage can be calculated as, 169mW·3ms = 507μJ.

In order to reduce the energy consumption, we scale the voltage to exploit the slack of communication tasks as shown in Fig. 5.2(c) and (d). To make the problem simple, we assume in this example that the task processing time of each on-chip module is fixed and known to us, i.e, the operating voltages of modules are known and given. Further, we assume that the supply voltages and the body bias voltages of all the drivers can be varied continuously in the ranges [1.1, 1.8]V and [-0.5, 0]V, respectively. In Fig. 5.2(c), communication task c_2 is scheduled with the supply voltage 1.4V and the body bias voltage -0.32V, respectively in order to exploit the slack of 1ms, while tasks c_1 and c_3 are scheduled with the nominal voltage because of zero slack. The total energy consumption of the communication tasks is 39mW·3ms+(87+87)mW·2ms = 465μJ, which is reduction in energy by 11$\%$ compared to the energy at the nominal voltages of 16-bit wide bus. In Fig. 5.2(d), the amount of slack is increased to 4ms by scheduling the communication tasks for the 32-bit wide bus. This slack is exploited by scaling the supply and body bias voltages of the communication tasks c_1 and c_3 to 1.2V and -0.39V, respectively, while c_2 is kept to the nominal voltage because there is no slack. In this case the CLTIs of all communication tasks do not overlap with each other, hence they can share a single bus. The total energy consumption is calculated as 174mW·1ms+ 2·56mW·2 = 398μJ, which corresponds to an energy reduction by 24$\%$ compared to the scheduling of Fig. 5.2(c).

It can be observed form the above example that while scheduling communication tasks for the synthesis of an on-chip communication architecture, more the available slack there is, the better are the results of synthesis in terms of on-chip communication resource sharing and energy consumption savings. Fig. 5.3 shows the plot of normalized amount of slack versus voltage scaling for different bus widths. It can be seen that the amount of slack increases with increasing bus width, but on the other side, we have to also pay the cost for chip size. In this thesis, we purpose a method to find the best trade-off for the synthesis of an on-chip communication architecture by simultaneously performing resource selection, scheduling, binding and voltage scaling of the communication bus.

5.1.2 Problem Formulation

We assume that a set of tasks have been partitioned and mapped onto the appropriate modules of an SoC. Each module m_i processes tasks and transfers the data to another module m_j, which has a data dependency with m_i. The data transfer from one module to another module takes place via a communication bus and this bus is driven by a driver of an on-chip module. We also assume that all the drivers are capable to scale the voltage during each data transfer. Since, the tasks have been efficiently mapped onto the modules, we further assume that the data processing time of each module is fixed

and given. Based on the mapped tasks, a directed acyclic extended graph $G_E(T, E)$ is obtained as shown in Fig. 5.1(b). The extended graph consists of two types of tasks, which are data processing tasks τ and communication tasks c. The data processing tasks τ are executed on the on-chip modules, while all tasks that use on-chip communication buses are called communication tasks c. For each task c_i its deadline dl_i, the data size to be transferred, and the switched capacitance C_{eff} are given. From the extended graph $G_E(T, E)$, the communication task graph $G_C(C, \Pi)$ is obtained with a start node S and a deadline node dl. In the communication task graph $G_C(C, \Pi)$, $c \in C$ be a set of communicating tasks and their data dependency between the communication tasks is defined by a set $\Pi \subseteq (C \times C)$, consisting of two-tuples (c_i, c_j) where a successor c_j depends on the results of the predecessor c_i. This data dependency between communication tasks is constrained by a set $Depn \subseteq (C \times C \times W)$ consisting of 3-tuples (c_i, c_j, w) such that $\forall i, j \in [1 \ldots N]$, $(c_i, c_j)_{i \neq j} \in \Pi | \Pi \subseteq C \times C$, a task c_j can start transferring data no earlier than w time units after the completion of it data transfer by c_i. The time constraint w is estimated using Eq. (4.1) for all tasks τ.

We assume that the supply voltage V_{dd} and the body bias voltage V_{bs} of each data processing task τ_i are known and provided to calculate the execution time of the task(s) in a module. Unlike this, the supply voltage V_{dd} and the body bias voltage V_{bs} of each communication task $c \in C$ are unknown and to be identified. We further assume that each task $c \in C$ can vary its supply voltage V_{dd} and body bias voltage V_{bs} within certain continuous ranges (for continuous voltage scaling), or within a set of discrete voltages $\{(V_{dd_z}, V_{bs_z})\}$ (for a discrete voltage scaling problem).

5.1.3 Complexity Analysis

Theorem 5.1.1 *The complexity of on-chip communication bus synthesis and continuous/discrete voltage scaling problem is **NP**-hard.*

Proof The discrete time-cost trade-off (DTCT) problem is known to be **NP**-hard [50], while its continuous variant, the linear time-cost trade-off (LTCT) problem can be solved in polynomial time complexity [66]. To make a better understanding of our problem, we give a new name to the on-chip communication bus synthesis and voltage scaling problem by "on-chip communication bus selection and voltage selection" problem. If we assume that only the supply voltage is scaled during the bus synthesis, our problem of discrete on-chip communication bus selection and continuous voltage selection is similar to the problem of DTCT. The discrete bus width selection and continuous voltage selection (DBS-CVS) changes duration of the CLTI and its energy consumption. This also applies for the discrete bus width selection and discrete voltage selection (DBS-DVS). Hence, DTCT $\in$ DBS-CVS and DTCT $\in$ DBS-DVS, which prove that DBS-CVS and DBS-DVS problems are **NP**-hard.

A linear relaxation method [145] is used to obtain an approximation algorithm of the discrete bus width selection and discrete voltage selection (DBS-DVS) problem. This method is used to get a lower bound on the value of the optimal solution such as voltage and bus width. Linear relaxation $\tilde{P}$ of a discrete problem P is a linear problem that consists of the same set of communication tasks c and processing tasks τ. The interval $[V_{l_c}^{\tilde{P}}, V_{u_c}^{\tilde{P}}]$ is given by $V_{l_c}^{\tilde{P}} := h_{l_c}^{P}$ and $V_{u_c}^{\tilde{P}} := k_{u_c}^{P}$ for each communication task $c \in C$. Where, $h_{l_c}^{P}$ and $k_{u_c}^{P}$ are the lower and upper voltage bounds before applying the linear relaxation to P. In the linear relaxation method, we transform arbitrary discrete problems to those problems with at most only few possible alternatives (voltage and bus width) for the duration of each CLTI of $c \in C$.

In this section, we consider discrete bus width selection and voltage selection problem for instances P of the l-DBS-DVS problem, for arbitrary $l \in \mathbb{N}$. For details of the approximation method, we refer the interested reader to [145]. The solutions of the problem can be obtained by computing an optimal realization of $\tilde{V}$ (voltages) and $\tilde{b_r}$ (bus width) of the linear relaxation $\tilde{P}$ of problem P and round them appropriately to a feasible realization of P. The quality of this realization can be tested by comparing its value, i.e., its deadline dl, to the value of the realization we started with. We call a realization $\tilde{V}$ of $\tilde{P}$ integral optimal for a deadline dl, if $\tilde{V}$ is the best integral realization of $\tilde{P}$ satisfying CLTI$(\tilde{V}) \leq dl$.

Theorem 5.1.2 *If $V \in \{V_1, V_2, \cdots, V_n\}$ and $b_r \in \{b_1, b_1, \cdots, b_n\}$ are integrals of voltage and bus width respectively, for all $c \in C$, then the LTCT solve algorithm [129] computes an integral optimal realization of deadline dl and bus width b_r in $O[|V| \cdot |b_r| \cdot |C|^2\ log|C|]$ time complexity.*

Proof In [129] an algorithm to solve the linear time-cost trade-off (LTCT) problem was proposed. The run time of each iteration of the algorithm is dominated by the run time needed to find the minimum cost and time. This can be done in $O[nm\ log(n^2/m)]$ time [66], where n denotes the number of vertices and m the number of edges of the communication task graph $G_C(C, \Pi)$. Since, there are no isolated vertices in the graph $G_C(C, \Pi)$, we get $n \leq m \leq |C|$, where $|C|$ is the cardinality of communication tasks. Hence, for the DBS-DVS problem with a discrete number of voltages $|V|$ and buses $|b_r|$, the overall run time of LTCT algorithm can be written as $O[|V| \cdot |b_r| \cdot |C|^2\ log|C|]$. The number of iterations in the LTCT solve algorithm depends on $|V|$ and $|b_r|$.

Theorem 5.1.3 *Let P be an instance of the l-DBS-DVS problem and $\tilde{V}$ be a realization of voltage for the linear relaxation $\tilde{P}$ of P.*

(a) If $\tilde{V}$ is an integral optimal for the deadline dl, then $CLTI^{\tilde{P}}(\tilde{V}) \leq dl^P$.

(b) If $\tilde{b_r}$ is an integral optimal for the deadline dl, then $CLTI^{\tilde{P}}(\tilde{b_r}) \leq dl^P$.

(c) The LTCT solve algorithm computes integral optimal realization of problem $\tilde{P}$ for all deadlines dl and buses b_r with a time complexity of $O(l_v \cdot l_b \cdot |C|^3\ log|C|)$.

Proof We are interested to realize an integral $\tilde{V}$ of DBS-DVS problem $\tilde{P}$ and all feasible realizations of P are integral, without loss of generality, we assume that the deadline *dl* in part (a) is integral too (because otherwise we can replace *dl* by $\lfloor dl \rfloor$). So, there exits an optimal, integral realization $\tilde{V}'$ and $\tilde{b}'$ for the integral deadline dl and the LTCT solve algorithm can be used to solve it. Similarly, part (b) can also be proved like part (a).

From part (a) and (b), we obtain integral optimal realizations by computing optimal realizations for integral deadlines *dl* and buses b_r. From Theorem 5.1.2 and 5.1.3, the DBS-DVS problem can be solved with a time complexity of $O(l_v \cdot l_b \cdot |C|^3 \; log|C|)$. The run time of the algorithm depends on the number of discrete voltages l_v, discrete buses l_b and C. For a fixed number of communication tasks c, the run time is proportional to the product of l_v and l_b. Hence, solving the DBS-DVS optimization problem has a *quasi-polynomial* time complexity.

5.2 Communication Bus Model

We assume that the supply voltage V_{dd} and the body bias voltage V_{bs} of each data processing task τ_i are provided to calculate the task execution time in a module. Furthermore, we consider the shared bus based communication architecture with driver and receiver, which are connected to bus and module as shown in Fig. 5.4. The architecture can have hierarchical buses, which in turn can be high speed buses (HSB) and/or low speed buses (LSB). These HSB and LSB buses are connected by the bridge to enhance the communication among the modules. The driver that initiates the data transfer between the modules, is capable to scale the voltage of each communication task dynamically in order to exploit the slack and to reduce total energy consumption. In between module and driver/receiver, there is an adapter, which converts the logic value of data values from modules and buses. For example, at certain instant t data is transfered between modules m_2 and m_1 with their supply voltages (V_{dd_2}, V_{dd_1}) and the bus voltage V_{dd}, the adapter of m_2 converts the logic values of data from V_{dd_2} to V_{dd} and the adapter of m_1 converts the logic value of data from V_{dd} to V_{dd_1}. The supply voltage of an adapter changes dynamically between two voltages of module and bus in a multiplexed manner.

The total power consumption of a bus based interconnect is given by both dynamic and leakage power consumption. The dynamic power consumption of a driver is due to the charging and discharging of the wire capacitance C_w, which is driven by an on-chip bus driver. This dynamic power of a driver is obtained as [56,107],

$$P_{dri_{dyn}} = \alpha_\tau \cdot f \cdot (C_{dri} + C_w) \cdot V_{dd}^2 \tag{5.7}$$

Where, α_τ is the switching activity of the communication tasks $c_i \in C$, f is the operating frequency of bus, C_{dri} and C_w are the capacitances of the drivers and wires,

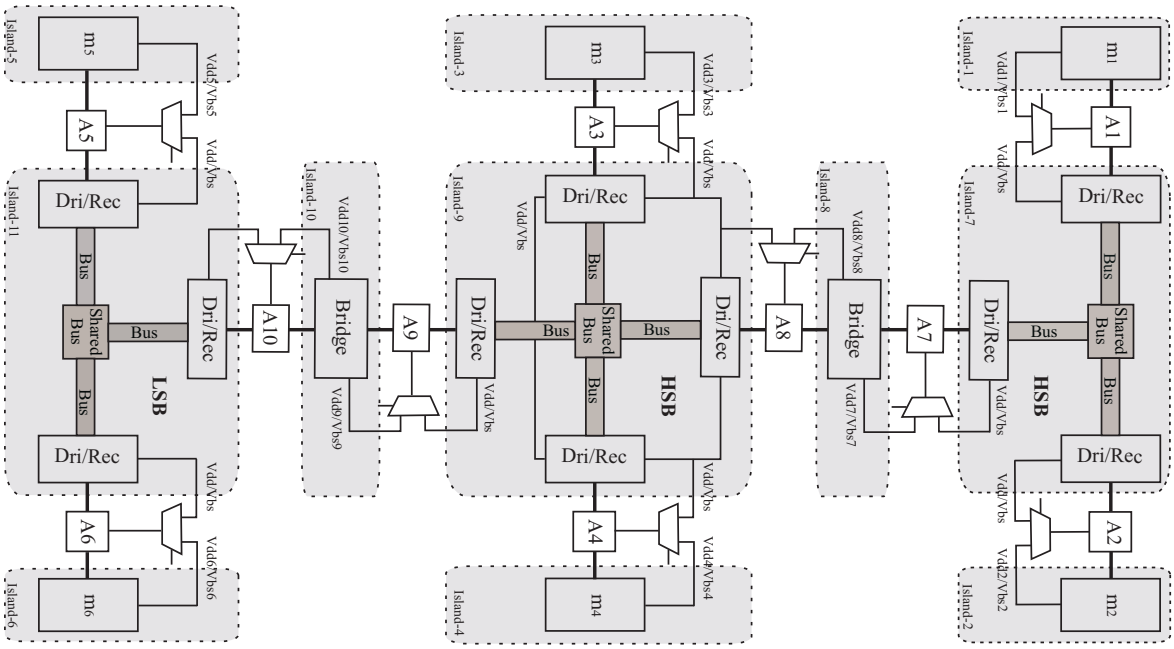


Fig. 5.4: On-chip communication architecture with voltage scalable driver and receiver

respectively.

Furthermore, the leakage power consumption of a driver can be expressed as [56, 107],

$$P_{dri_{leak}} = L_g \cdot (V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju}) \tag{5.8}$$

Where V_{bs} is the body bias voltage and I_{Ju} represents the body junction leakage current. The fitting parameters K_3, K_4 and K_5 denote technology dependent constants and L_g reflects the number of gates. Similarly, the dynamic power consumption of a receiver can be expressed as [56,107],

$$P_{rec_{dyn}} = \alpha_\tau \cdot f \cdot C_{rec} \cdot V_{dd}^2 \tag{5.9}$$

Where C_{rec} is the capacitance of receiver. The leakage power consumption of a receiver is expressed as,

$$P_{rec_{leak}} = L_g \cdot (V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju}) \tag{5.10}$$

In our method of on-chip communication synthesis, we evaluate the total energy consumption of the communication tasks c for a bus with different widths.

5.3 Combined Bus Synthesis and Supply Voltage Scaling

5.3.1 Continuous Voltage Scaling

Problem 5.3.1.1 *(Combined scheduling, continuous supply voltage scaling, bus selection and binding of communication tasks $c \in C$ to minimize the communication bus width and the number of buses with reduced communication energy under the constraints of supply voltage and the real-time constraints) Perform simultaneous scheduling, supply voltage scaling, bus selection and binding of communication tasks that minimize $\sum_{r \in R} Cost_r \cdot b_r$, where $r \in R$ is a library of buses with different bus widths and the variable b_r is the optimization variable; subject to: $\sum_{c \in C}(dl_c - t - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}}) \geq 0$ and $V_{dd_{min}} \leq V_{dd} \leq V_{dd_{max}}$, for all $t \in \{0, \cdots, \lambda\}$, where λ is the maximum possible time to schedule communication tasks $c \in C$, $CLTI_{c,r,V_{dd}}$ is the communication lifetime interval, which is a function of width r of bus b and the supply voltage V_{dd}.*

The nonlinear programming formulation for the simultaneously scheduling, voltage scaling, bus selection, and binding of communication tasks c is given as follows:

Minimize:

$$\sum_{r \in R} Cost_r \cdot b_r \tag{5.11}$$

Where, $r \in R$ is a library of on-chip communication buses of different bus widths, for example, buses of 16, 20, 24, $\cdots$, 128-bit. The $Cost_r$ of bus type r is expressed in terms of bus width, like the cost of a 32-bit wide bus is double the cost of a 16-bit wide bus. The bus costs are stored in a lookup table for each bus width. The objective is to minimize the total cost of the buses by maximizing bus sharing among the communication tasks. The variable b_r is an optimization variable of Eq. (5.11).

subject to,

$$\forall c \in C, \sum_{r \in R} \sum_{t=ASAP_c}^{\Psi} X_{c,t,r,V_{dd}} = 1 \tag{5.12}$$

$$\Psi = (ALAP_c + d_{min_c} - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}}) \tag{5.13}$$

Eq. (5.12) defines a binding constraint for the simultaneous bus synthesis and supply voltage scaling problem, where each communication task $c \in C$ must be mapped to a single bus with bus width r, operating at a single time instant t, with a supply voltage V_{dd}. The binary decision variable $X_{c,t,r,V_{dd}} \in \{0, 1\}$, indicates scheduling of a communication task $c \in C$ at time $t \in \{0, \cdots, \lambda\}$, with bus width r and supply voltage V_{dd}, respectively. The term λ is the maximum possible time to schedule a task $c \in C$, and $\delta_{i,j}^{\Delta V_{dd}}$ is the time overhead delay due to switching of the voltage from V_{dd_i} to V_{dd_j}. In

Eq. (5.13) the term Ψ gives an amount of slack for each communication task $c \in C$ and this slack is proportional to $CLTI_{c,r,V_{dd}}$ and $\delta_{i,j}^{\Delta V_{dd}}$. A communication task $c \in C$ with ALAP time $ALAP_c$ cannot be executed later than Ψ, when data is transferred through a bus b_r with a data transfer duration $CLTI_{c,r,V_{dd}}$ of a task. Where, the constant d_{min_c} is the minimum time to execute a communication task $c \in C$ to meet the deadline dl_c. In Eq. (5.12), for each bus width r and for each supply voltage V_{dd}, the amount of slack changes for each communication task c.

In Eq. (5.14), we introduce a set Ω, which represents the set of all time instants that any communication task could possibly start at,

$$\Omega = \bigcup_{c \in C} \{ASAP_c, \cdots, ALAP_c\} \tag{5.14}$$

$$\begin{aligned} &\forall t \in \Omega, \forall r \in R, \\ &\sum_{c \in C} \sum_{\substack{(t' \in \{t, \cdots, t+d_r-1\} \\ \cap \{ASAP_c, \cdots, \psi\})}} X_{c,t',r,V_{dd}} \leq b_r \end{aligned} \tag{5.15}$$

The fact is that no communication bus b with width r can execute more than one communication task at an instant t with supply voltage V_{dd}, is expressed as a constraint in Eq. (5.15). The first sum is over all communication tasks with bus width r, the second sum is over a "time window" covering all start times $t^{'}$ for which communication tasks could overlaps.

$$\begin{aligned} &\forall\, (c^{'}, c) \in \Pi,\ \forall\, (c^{'}, c, w) \in Depn, \\ &\sum_{r \in R} \sum_{\substack{t= \\ ASAP_c}}^{\Psi} t \cdot X_{c,t,r,V_{dd}} \geq \\ &\sum_{r \in R} \sum_{\substack{t'= \\ ASAP_{c'}}}^{\Psi'} (t^{'} + CLTI_{c',r,V_{dd}} + w + \delta_{i,j}^{\Delta V_{dd}}) \cdot X_{c',t',r,V_{dd}} \end{aligned} \tag{5.16}$$

$$\Psi^{'} = (ALAP_{c'} + d_{min_c^{'}} - CLTI_{c',r,V} - \delta_{i,j}^{\Delta V_{dd}}) \tag{5.17}$$

The data dependency between communication tasks is expressed as Eq. (5.16). The term on the right hand side of the equation expresses a predecessor task $c^{'}$, while the term on left hand side of the equation expresses a successor task c, which should be executed only after the execution of the task $c^{'}$. In Eq. (5.16), the first sum is over all communication tasks $c \in C$ with bus width r and the second sum is for a start time t with its possible time that ranges from $ASAP_c$ to Ψ (amount of slack for each successor task c as shown in Eq. 5.13). The delay w is the delay to execute the data processing task τ between two communication tasks and this delay can be evaluated using Eq.

(4.1). Eq. (5.17) gives an amount of slack for each predecessor task c and the slack is a function of the bus width b_r and supply voltage V_{dd}; this is same as for a successor task c shown in Eq. (5.13).

$$V_{dd_{min}} \leq V_{dd} \leq V_{dd_{max}} \tag{5.18}$$

The slack of each communication task $c \in C$ is exploited to share a communication bus and reduce the total energy consumption by scaling the supply voltage V_{dd} for each communication task from the nominal voltage. The voltage is scaled continuously between two upper and lower bound $V_{dd_{max}}$ and $V_{dd_{min}}$, respectively. This continuous voltage scaling is constrained by Eq. (5.18).

$$\forall\, c \in C, \sum_{r \in R} \sum_{\substack{t= \\ ASAP_c}}^{\Psi} ((dl_c - t - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}}) \cdot X_{c,t,r,V_{dd}}) \geq 0 \tag{5.19}$$

The sum of a deadline dl_c, a start time t, the data transfer time $CLTI_{c,r,V_{dd}}$ and the delay overhead due to voltage switching $\delta_{i,j}^{\Delta V_{dd}}$ of each communication task $c \in C$ should be greater than or equal to zero for an on-chip bus b with width r and supply voltage V_{dd} as shown in Eq. (5.19). Where, the first sum is over all communication tasks $c \in C$ taken into account for different bus widths b_r, the second sum includes all possible start times t, which range from $ASAP_c$ to the slack of the bus b of width r and supply voltage V_{dd}.

The above communication bus optimization problem has linear objective function and nonlinear constraints. This is relaxed as a convex quadratic optimization problem and can be solved using any convex optimization tool.

5.3.2 Discrete Voltage Scaling

Problem 5.3.2.1 *(Combined scheduling, discrete voltage scaling, bus selection, and binding of communication tasks $c \in C$, to minimize communication the bus width and the number of buses with reduced communication energy) Perform scheduling, discrete voltage scaling, bus selection, and binding of communication task $c \in C$ to minimize communication bus cost (see Eq. (5.11)); subject to: $\sum_{c \in C}(dl_c - t - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}}) \geq 0$ and a discrete set of supply voltage $V_{dd} \in \{V_{dd_1}, V_{dd_2}, \cdots, V_{dd_z}\}$.*

As digital system designs are most often restricted to a finite set of discrete performance modes, it is not possible to apply continuously selected voltages to them. In **Sec.** 5.1.3, we have demonstrated that both DBS-CVS and DBS-DVS problems are **NP**-hard. In this subsection, a heuristic method is presented to transform a continuously selected optimal supply voltage V_{dd}^{opt} using the NLP formulation given in **Sec.** 5.4.2.1, into a discrete set of supply voltage $V_{dd}^{'}$. The heuristic takes the optimal bus width b_r^{opt}

```
HEURISTIC-DBS-DVS(b_r^opt, V_dd^opt)
 1  V_dd_z ← GETLIBOFDISCRETEVDD();
 2  /*Linear relaxation method*/
 3  V_dd^' ← ⌈V_dd^opt⌉ if V_dd^opt ∈ [V_dd_z^LB, V_dd_z^UB];
 4  /*Check condition*/
 5  for c ∈ C and c^' ∈ C
 6  do
 7     if (t + CLTI_{c,r,V_dd^'} + δ_{i,j}^{ΔV_dd^'} ≤ dl)
 8       then
 9     return V_dd^';
10      else
11     (V_dd^') ← GETNEXTVALUE();
```

Algorithm 5.1: Heuristic for discrete supply voltage selection.

and supply voltage V_{dd}^{opt} as inputs and finds a near-optimal supply voltage as shown in Algorithm 5.1. These optimal values are obtained using the formulation presented in **Sec.** 5.4.2.1, where the selection of supply voltage is performed continuously. At line 1 of Algorithm 5.1, a library of discrete set of supply voltages V_{dd_z} is read. Since the continuously selected optimal supply voltage is not feasible for practical implementation, thus at line 3, the upper bound of V_{dd}^{opt} is selected, which must be an element of V_{dd_z}. We could choose the lower bound of supply voltage to get the minimum energy consumption, however, this may violate the given real-time constraints. At line 4-8 of algorithm, the overall delay is checked with a deadline dl_c for the bus width b_r^{opt} and the supply voltage $V_{dd}^{'}$. If the condition meets then the heuristic returns those near-optimal values at line 9. Otherwise, at each time next supply voltage, which is greater than the $V_{dd}^{'}$ is selected from the library of discrete supply voltages V_{dd_z} at line 11 and the condition is checked again at line 4-8 for a new supply voltage.

5.4 Extension to Body Biasing

In **Sec.** 5.3, we have discussed the optimization problem of simultaneous on-chip communication bus synthesis and supply voltage scaling, which finds the optimal bus width and the number of buses with reduced on-chip communication energy. Intuitively from Eq. (5.4) the dynamic power consumption is a nonlinear function of the supply voltage, which is an effective means to reduce dynamic power consumption of an on-chip communication bus. However, the supply voltage can not be scaled down beyond a certain limit under a given real-time constraint. In this subsection, we investigate simultaneous supply voltage scaling and body biasing problem during the synthesis of on-chip communication buses.

While scaling the supply voltage for energy reduction, possible malfunction at low supply voltages appears to be a vulnerable to a problem with respect to the system performance. A common condition for a logic gate being properly functioning is a gain[1], which should be significantly larger than one in order to guarantee a level restoration from one stage to the next. The ultimate goal is to have per-stage gains large enough to keep the logic levels at all circuits nodes within the specified noise margins. It was shown theoretically that this condition can be satisfied even at extremely low supply voltages of a few hundred millivolts [104, 108, 151]. First practical examples of circuits operating at supply voltage as low as 0.2V was published in [152], however, for a sub-nanometer scaled CMOS technology, the above demonstration does not valid anymore. Thus, the possibility of a malfunction due to the low supply voltages has become a real challenge for future sub nanometer nodes. Furthermore, practical limits of voltage scaling arise from given real-time constraints of a system. It can be seen in Eq. (4.2) that the gate delay T_d and hence, the performance of a circuit apparently degrade, if the supply voltage is reduced while the threshold voltage is kept at the same level. From Eq. (5.8), low threshold voltages[2] cause excessively high leakage power consumption due to the leakage current. Hence, the aggressive supply and threshold voltage scaling eventually leads to the minimum total power consumption P_t (dynamic and leakage), which is defined an optimal pair of supply and threshold voltage values.

In practice, the optimum is usually not well defined because of unavoidable supply and threshold voltage uncertainties that can be due to temperature and process variability, short channel effects, and non-ideal supply voltage regulation. These voltage uncertainties result in delay and power variations and must be taken into account when determining the nominal voltage values. All of these issues are addressed in detail in **Chap.** 6.

5.4.1 Power Delay Analysis w.r.t Supply and Body Bias Voltages

In this subsection, the rate of change of power consumption and delay with respect to supply and body bias voltages are calculated. The goal is to analyze the individual contribution of supply and body bias voltage to power and delay, respectively. From the above Eqs. (5.7) and (5.8), the total power consumption (dynamic and leakage power) of an on-chip communication bus can be written as,

$$P_t = f \cdot C_{eff} \cdot V_{dd}^2 + V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju} \tag{5.20}$$

[1]The gain is the absolute value of the slope of the voltage transfer characteristics

[2]threshold voltage is a function of body bias voltage

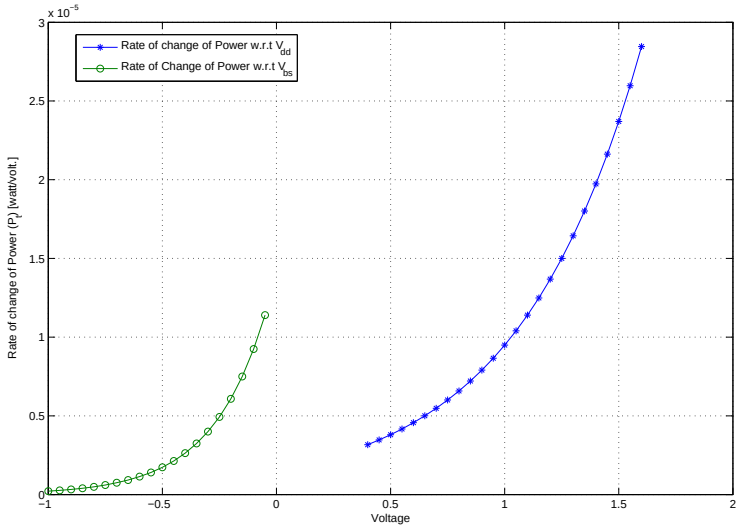


Fig. 5.5: Rate of change of power with respect to supply voltage V_{dd} and body bias voltage V_{bs}

The partial derivative of total power P_t with respect to supply voltage V_{dd} and body bias voltage V_{bs} can be obtained as,

$$\frac{\partial P_t}{\partial V_{dd}} = 2f \cdot C_{eff} \cdot V_{dd} + K_3 \cdot K_4 \cdot V_{dd} \cdot e^{K_4 V_{dd} + K_5 V_{bs}} + K_3 e^{K_4 V_{dd} + K_5 V_{bs}} \tag{5.21}$$

$$\frac{\partial P_t}{\partial V_{bs}} = K_3 \cdot K_5 \cdot V_{dd} \cdot e^{K_4 V_{dd} + K_5 V_{bs}} + I_{Ju} \tag{5.22}$$

Similarly, partial derivatives of the gate delay T_d with respect to supply and body bias voltages can be calculated as,

$$\frac{\partial T_d}{\partial V_{dd}} = \frac{K_6 - 2(1 + K_1)K_6 V_{dd}}{[(1 + K_1)V_{dd} + K_2 V_{bs} - V_{th}]} \tag{5.23}$$

$$\frac{\partial T_d}{\partial V_{bs}} = \frac{-2K_2 K_6 V_{dd}}{[(1 + K_1)V_{dd} + K_2 V_{bs} - V_{th}]} \tag{5.24}$$

Fig. 5.5 and 5.6 show the rate of change of power and delay with respect to both supply and body bias voltages. These are obtained for 70nm technology and the technology dependent parameters were extracted from [107, 2]. Where, K_1 = 0.063, K_2 = 0.153, K_3 = 5.38e-07, K_4 = 1.83, K_5 = 4.19, f = 15.6 GHz, K_6 = 5.26e-12, K_7 = -0.144, C_{eff} = 2.0e-15 F, I_j = 4.8e-10 A, and V_{th} = 0.244 V. Power and delay with respect to supply and body bias voltages are plotted for V_{dd} = 0.4 to 1.6V and V_{bs} = -0.05 to -1.0V, respectively. In Fig. 5.5, power with respect to supply voltage contributes to the dynamic power consumption, while the body bias voltage contributes to the leakage

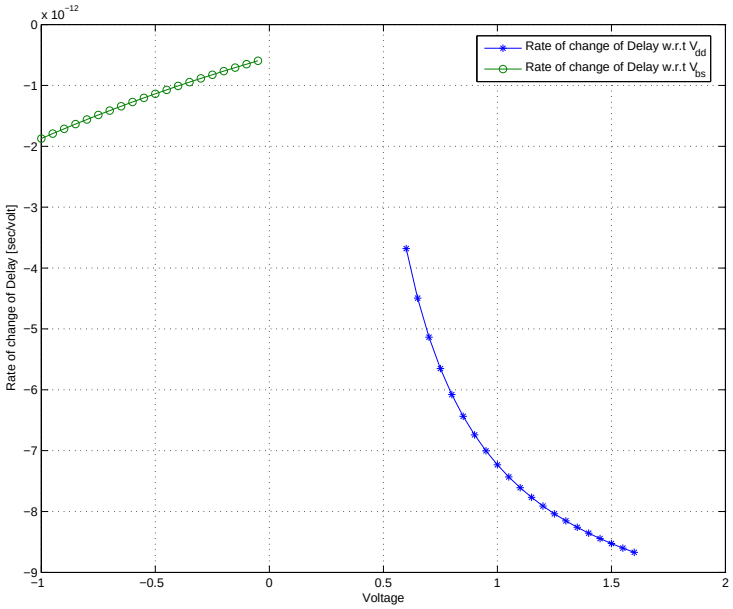


Fig. 5.6: Rate of change of delay with respect to supply voltage V_{dd} and body bias voltage V_{bs}

power from the total power consumption P_t and they are an exponential function of supply voltage V_{dd} and body bias voltage V_{bs}. It can be observed that rate of change of the leakage power is less than that of dynamic power and their values tend to be zero for voltages close to infinity. Furthermore, the leakage power will not reduce significantly, if body bias voltage is scaled below -1.0V, thus, -1.0V can be set as the lowest boundary of V_{bs} for the above considered example.

In Fig. 5.6, the contribution of supply voltage V_{dd} to the delay T_d is high incomparison to the contribution of body bias voltage V_{bs}. Where rate of change of delay T_d is almost linear with the body bias voltage V_{bs} and is an inverse function of supply voltage V_{dd}. For simultaneous supply voltage scaling and body biasing during the synthesis of on-chip communication bus, the supply voltage can be scaled to the minimum possible level in order to reduce power consumption, however, it may cause the violation of real-time constraints because the delay is an inverse function of the supply voltage. In contrast to the dynamic power, the leakage power is an exponential function of body bias voltage, while the delay is approximately linear with the body bias voltage. Hence, it is worth to scale the body bias voltage to the minimum level while the rest of the available slack is exploited to scale the supply voltage, which decreases the dynamic power consumption.

5.4.2 Continuous Voltage Scaling

Problem 5.4.2.1 *(Simultaneous scheduling, continuous voltages (supply and body bias) scaling, bus selection, and binding of communication task $c \in C$; to minimize bus width and the number of buses in order to reduce communication energy) Perform combined scheduling, supply/body bias voltage scaling, bus selection, and binding of communication tasks $c \in C$, which minimize communication cost $\sum_{c \in C} Cost_r \cdot b_r$ and reduce energy consumption, where R is a library of communication buses with different bus widths r; subject to: the real time constraint $\sum_{c \in C}(dl_c - t - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}}) \geq 0$ for all time $t \in \{0, \cdots, \lambda\}$, supply voltage constraint $V_{dd_{min}} \leq V_{dd} \leq V_{dd_{max}}$ and body bias voltage constraint $V_{bs_{min}} \leq V_{bs} \leq V_{bs_{max}}$ for continuous voltage scaling. λ is the maximum latest possible start time of communication tasks, $CLTI_{c,r,V_{dd},V_{bs}}$ is the communication lifetime interval of communication task c and it is a function of the bus width b_r, supply voltage V_{dd}, and body bias voltage V_{bs}; $\delta_{i,j}^{\Delta V_{bs}}$ is the delay due to switching of voltages from one level to another level and dl_c is the deadline of communication task $c \in C$.*

Above Problem 5.4.2.1 is an optimization problem with a set of discrete bus widths as a variable and continuous supply and body bias voltages with their upper and lower bounds as constraints. In this subsection, an optimization model is presented to synthesize an optimal bus width and the number of buses with reduced communication energy. The slack is exploited to maximize the sharing of on-chip communication buses and to reduce energy consumption by simultaneously scaling supply and body bias voltages during the synthesis of the on-chip communication bus. This means that the slack is exploited to maximize the sharing of buses and then, the voltages are scaled in order to reduce the communication energy only when there is slack. If there is no more slack left, the supply and body bias voltages are kept at their nominal values. As discussed in **Sec.** 5.4.1, with a simple example for 70nm CMOS technology, supply voltage has a higher contribution to the total power than the body bias voltage as depicted in Fig. 5.5. Furthermore, rate of change of delay due to body bias voltage is linear, while for supply voltage it is an inverse function as shown in Fig. 5.6. Hence, the delay characteristics of supply and body bias voltages are not similar, in the following formulation of simultaneous supply and body bias voltage scaling. First, body bias voltage is scaled to the minimum possible level and second, the supply voltage is scaled only if there is still some slack, else, the voltage is not scaled. This approach has two main advantages, first the leakage power can be reduced to the minimum level and second the delay does not change that much as it changes for supply voltage. As reported in several research results, the leakage power will be more vulnerable than dynamic power in sub-nanometer CMOS technology. Therefore, it is worth to exploit the slack first for the body bias voltage scaling then for supply voltage scaling.

In the following formulation for simultaneous on-chip communication bus synthe-

sis and continuous supply/body bias voltages scaling, we introduce first a nonlinear programming (NLP) model, which performs scheduling, continuous supply and body bias voltage scaling, bus selection, and binding of communication tasks. Since, data transfer delay CLTI is an inverse function of bus width and voltages, the on-chip communication synthesis problem is a nonlinear programming (NLP) problem, which is relaxed to the convex quadratic optimization as follows:

Minimize: The objective function of Problem 5.4.2.1 is similar to Problem 5.3.1.1 and it is shown in Eq. (5.11). The communication bus cost is expressed in terms of bus width. The objective is to minimize the total cost due to the communication buses by maximizing bus sharing among the communication tasks. The optimization variable is b_r (bus b with width r).

subject to,

$$\forall c \in C, \sum_{r \in R} \sum_{t=ASAP_c}^{\Psi} X_{c,t,r,V_{dd},V_{bs}} = 1 \tag{5.25}$$

$$\Psi = (ALAP_c + d_{min_c} - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}}) \tag{5.26}$$

Eq. (5.25) gives a binding constraint for each communication task $c \in C$, which is mapped to operating time t, a single communication bus b with width r, supply voltage V_{dd} and body bias voltage V_{bs}. The first sum is over all communication tasks $c \in C$ for different bus widths b_r, the second sum is for possible start time t for data transformations using bus b_r. The possible start time t ranges from $ASAP_c$ to $ASAP_c$ + slack. The *ASAP* time of a communication task is always fixed, but, the amount of slack time varies with bus width b_r, supply voltage V_{dd}, and body bias voltage V_{bs} as shown in Eq. (5.26). In the equation, $ALAP_c$ and the minimum amount of delay d_{min_c} to execute a communication task c are fixed[3], other terms such as data transfer duration $CLTI_{c,r,V_{dd},V_{bs}}$, delay overhead due to supply voltage switching $\delta_{i,j}^{\Delta V_{dd}}$, and delay overhead due to body bias voltage switching $\delta_{i,j}^{\Delta V_{bs}}$ are variables, which change when one of the variables bus width b_r, supply voltage V_{dd} or body bias voltage V_{bs} change. The binary decision variable $X_{c,t,r,V_{dd},V_{bs}} \in \{0,1\}$, defines the scheduling of a communication task c at time $t \in \{0, \cdots, \lambda\}$, the bus b with width r, the supply voltage V_{dd}, and the body bias voltage V_{bs}. In Eq. (5.25) body bias voltage V_{bs} is scaled first to the lowest possible level and then the supply voltage V_{dd} is scaled if there still remains the slack. Otherwise supply voltage V_{dd} is kept to its nominal value.

Like just as in Problem 5.3.1.1, we introduce a set Ω that represents the set of all time instants at which any communication task could possibly start to transfer data using a

[3]the slack of data processing tasks τ is exploited in order to reduce energy consumption. Thus its delay equal to the deadline

bus b_r as shown in Eq. (5.14),

$$\forall t \in \Omega, \forall r \in R, \\ \sum_{c \in C} \sum_{\substack{(t' \in \{t, \cdots, t+d_r-1\} \\ \cap \{ASAP_c, \cdots, \psi\})}} X_{c,t',r,V_{dd},V_{bs}} \leq b_r \tag{5.27}$$

Eq. (5.27) defines the resource constraint for simultaneous communication bus synthesis, supply and body bias voltage scaling such that each communication bus b with bus width r cannot execute more than one communication task at time t with supply voltage V_{dd} and body bias voltage V_{bs}. The first sum is over all communication tasks with bus width r and the second sum is over a "time window" covering all start times $t^{'}$ for which the communication tasks could overlap. If two communication tasks c_i and c_j use the same communication bus b with width r and their communication lifetime interval CLTIs overlap then a variable b_r (bus b with bus width r) will increase by 1. This means that there is a conflict between two communication tasks and consequently two separate buses are needed to transfer data. Since the primary goal of the optimization model is to minimize communication cost, the algorithm tries to reduce the conflict among the communication tasks keeping the supply and body bias voltages close to their nominal voltages.

$$\forall\, (c^{'}, c) \in \Pi,\ \forall\, (c^{'}, c, w) \in Depn, \\ \sum_{r \in R} \sum_{\substack{t= \\ ASAP_c}}^{\Psi} t \cdot X_{c,t,r,V_{dd},V_{bs}} \geq \\ \sum_{r \in R} \sum_{\substack{t'= \\ ASAP_{c'}}}^{\Psi'} (t^{'} + CLTI_{c',r,V_{dd},V_{bs}} + w + \delta_{i,j}^{\Delta V_{dd}} + \delta_{i,j}^{\Delta V_{bs}}) \cdot X_{c',t',r,V_{dd},V_{bs}} \tag{5.28}$$

$$\Psi^{'} = (ALAP_{c'} + d_{min_c^{'}} - CLTI_{c',r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}}) \tag{5.29}$$

Eq. (5.28) defines a dependency constraint between two tasks $(c, c^{'}) \in Depn$, where task c is a successor and task $c^{'}$ is a predecessor such that task c should not be executed before time $(t + CLTI_{c',r,V_{dd},V_{bs}} + w + \delta_{i,j}^{\Delta V_{dd}} + \delta_{i,j}^{\Delta V_{bs}})$, which is the delay to transfer data by task $c^{'}$ with its bus of width r, supply and body bias voltage V_{dd} and V_{bs}, respectively. The delay w is time delay to execute data processing task τ between two communication tasks and this delay can be evaluated using Eq. (4.1).

$$V_{bs_{min}} \leq V_{bs} \leq V_{bs_{max}} \tag{5.30}$$

Each time the slack of communication task $c \in C$ is exploited to share communication buses and reduce the total energy consumption by scaling supply voltage V_{dd} and body

bias voltage V_{bs} for each communication task from their nominal voltages. While scaling the voltages, the first priority is given to body bias voltage to reduce the leakage power and the second priority is given to supply voltage only if slack is available. Eqs. (5.18) and (5.30) define the constraint to scale voltages for both supply and body bias, respectively.

$$\forall\, c \in C,$$
$$\sum_{r \in R} \sum_{\substack{t= \\ ASAP_c}}^{\Psi} ((dl_c - t - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}}) \cdot X_{c,t,r,V_{dd},V_{bs}}) \geq 0 \tag{5.31}$$

Eq. (5.31) checks the violation of overall delay with a given deadline of communication tasks dl_c. The summation of deadline dl_c, start time t, the data transfer time $CLTI_{c,r,V_{dd},V_{bs}}$, and the delay overhead due to supply and body bias voltages switching $\delta_{i,j}^{\Delta V_{dd}}$ and $\delta_{i,j}^{\Delta V_{bs}}$ should be greater than or equal to zero for an on-chip bus b with width r, the supply voltage V_{dd}, and the body bias voltage V_{bs}.

5.4.3 Discrete Voltage Scaling

Problem 5.4.3.1 *(Simultaneous scheduling, discrete supply and body bias voltage scaling, bus selection and binding of communication tasks $c \in C$ to minimize bus width and number of buses with reduced communication energy) Perform combined scheduling, discrete supply and body bias voltage scaling, bus selection and binding of communication of tasks $c \in C$ to minimize communication cost (see Eq. (5.11)) subject to, the real-time constraints (see Eq. (5.31)) and a set of discrete supply and body bias voltages $\{(V_{dd_1}, V_{bs_1}), \cdots, (V_{dd_z}, V_{bs_z})\}$. Furthermore, $\delta_{i,j}^{\Delta V_{dd}}$ and $\delta_{i,j}^{\Delta V_{bs}}$ are delay overhead due to switching of supply and body bias voltages, respectively.*

The formulation of Problem 5.4.3.1 is similar to Problem 5.3.2.1 except for the body bias voltage scaling technique. A heuristic for combined discrete supply and body bias voltages selection technique is presented in Algorithm 5.2. It takes an optimal bus width b_r^{opt}, supply voltage V_{dd}^{opt}, and body bias voltage V_{bs}^{opt} as inputs and transforms the continuously selected voltages into a discrete set. Similar to Algorithm 5.1, first discrete sets of voltage are read at line 1 and 2 for supply and body bias voltages, respectively. Using a linear relaxation method, at line 4 and 5, corresponding voltages are quantized to their upper bound. At line 9, the condition for real-time constraints is checked for the selected bus width, supply, and body bias voltages. If the condition is satisfied the selected discrete set of voltages are returned at line 12. Otherwise, the iteration is repeated and the next higher value of supply voltage is selected at line 14. The voltage of body bias is kept as before once it is quantized.

HEURISTIC-DBS-DVS($b_r^{opt}, V_{dd}^{opt}, V_{bs}^{opt}$)
1 $V_{dd_z} \leftarrow$ GETLIBOFDISCRETEVDD();
2 $V_{bs_z} \leftarrow$ GETDISCRETEVBS();
3 /*Linear relaxation method*/
4 $V_{dd}^{'} \leftarrow \lceil V_{dd}^{opt} \rceil$ **if** $V_{dd}^{opt} \in [V_{dd_z}^{LB}, V_{dd_z}^{UB}]$;
5 $V_{bs}^{'} \leftarrow \lceil V_{bs}^{opt} \rceil$ **if** $V_{bs}^{opt} \in [V_{bs_z}^{LB}, V_{bs_z}^{UB}]$;
6 /*Check condition*/
7 **for** $c \in C$ **and** $c^{'} \in C$
8 **do**
9 **if** $(t + CLTI_{c,r,V_{dd}^{'},V_{bs}^{'}} + \delta_{i,j}^{\Delta V_{dd}^{'}} + \delta_{i,j}^{\Delta V_{bs}^{'}} \leq dl)$
10 **then**
11
12 **return** $(V_{dd}^{'}, V_{bs}^{'})$;
13 **else**
14 $(V_{dd}^{'}) \leftarrow$ GETNEXTVALUE();

Algorithm 5.2: Heuristic for discrete supply voltage selection.

5.5 Summary

As we discussed in **Chap.** 2 that, the device and wire geometry scaling trends have a significant impact on the delay and the power consumption of the wires. Furthermore, due to the cramming more and more numbers of transistors in a single chip, power consumption per unit area is increasing. As a result of this, thermal effects and noise become challenges and that degrade circuit performance and reliability. Thus power optimization of a system has become increasingly important at each level of abstraction. Previously, numerous research works have contributed to optimize the power consumption of a system. They are mainly at layout level, circuit level, and post synthesis architecture level. At post synthesis architecture level, voltage scaling and bus encoding techniques [148,149,20] were applied to reduce the power consumption, and these techniques are used most commonly in industry and academia. In a real-time distributed embedded system, workload offer to the system is not uniform over time, thus slack of each task can be exploited by scaling supply and body bias voltages [162, 37, 57, 35]. Furthermore in [16, 17] dynamic supply voltage scaling and adaptive body biasing techniques were proposed to reduce the power consumption of fat wires and repeater based communication buses. All the above approaches assume that the bus width, the number of buses, and the communication bus topology have been previously identified and are given.

The main contribution of this chapter is to perform the simultaneous communication bus synthesis and voltage scaling in order to find a trade-off between energy consumption and communication cost. Since the slack of each communication task

is a function of the bus width and voltage pair (supply and body bias), we exploited this slack by sharing the communication bus and reducing its energy consumption. The problem of simultaneous communication bus synthesis and voltage scaling is a nonlinear optimization problem, which was relaxed as a convex quadratic optimization problem and solved efficiently using a convex optimization tool. We presented two separate models, supply voltage scaling and its extension to body biasing for both continuous and discrete voltage constraints. The optimization algorithm takes a set of communication tasks, a library of bus width, and voltage constraints with upper and lower bounds as inputs and finds an optimal bus width and number of buses. Though scaling of supply voltage reduces the dynamic power consumption quadratically, it can not be scaled to very low level due its effect on noise and circuit reliability. Thus, we presented an extended model which performs combined supply voltage scaling and body biasing to reduce both dynamic and leakage power consumption. In this model, first, we analyzed an individual contribution of supply and body bias voltage to power and delay, respectively. The analysis showed that the contribution of supply voltage to delay is high in comparison to body bias voltage, while the contribution to power consumption of the supply voltage is less than the body bias voltage. Hence for a combined supply and body bias voltage scaling method, it is better to first scale body bias voltage to the minimum level and the rest of available slack is exploited to scale the supply voltage, which decreases the dynamic power consumption.

Although the continuous voltage scaling technique would result in a better run time complexity and energy consumption than discrete voltage scaling, it can not be used for a digital system design due to practical limitations. Further, we proved in **Sec.** 5.1.3 that the discrete bus selection and discrete voltage scaling (DBS-DVS) problem is **NP**-hard. Therefore, we proposed a heuristic that performs discrete voltage scaling of communication tasks in a polynomial time complexity. In the second part of experiments, we performed simultaneous communication bus synthesis and discrete voltage scaling with power and delay overhead due to voltage switching. Experimentally, we found that the reduction in energy consumption due to discrete voltage scaling is greater than continuous voltage scaling.

Chapter 6

Simultaneous Bus Synthesis and Voltage Scaling Under Variations

Contents

In **Chap.** 4, we proposed a method to synthesize buses under real-time constraints. After applying an optimization to the bus synthesis problem, the algorithm finds the optimal bus width and the number of buses. However, there is still a significant of slack left for the synthesized optimal bus width. This indicates the underutilization of communication resources. To cope with this problem, in **Chap.** 5, we presented an energy

aware communication bus synthesis technique, which exploits the slack of communication tasks by scaling the voltage and results in reduction of communication energy consumption. So far, in previous chapters, we assumed that data size to be transferred by each communication task is fixed. However, this is not the case in a real-time embedded system that runs a diversity of applications, which results in a non uniform workload to the system. Thus a communication bus that is synthesized without taking into account the systems peak load can turn a major performance bottleneck. In the past quite a few efforts have been made to consider variable systems loads. In [96] Lahiri et al. propose an intermodule communication statistics method, which profiles several applications at system level and selects the one communication model with a peak load. This peak load is considered as the worst case scenario and later they refine the communication bus architecture. However, if load offers to a system is normal then under typical load conditions the bus will be underutilized. In [166] Varatkar et al. propose an analytical method to model variable data traffic. The method is based on the markov chain producer and consumer model and synthesizes the buffer size at each interface. However, the method is targeted for network-on-chip architectures, which is not the scope of our work.

The presented work in this chapter makes two contributions. First, it proposes an analytical method to model variations in data size of each communication task. For simplicity, only supply voltage is scaled in order to reduce energy consumption, however, the body biasing technique can be integrated in our model without any major change. Based on this model, the delay constraint of each task is formulated as a probabilistic nonlinear constraint, which is later casted into a deterministic constraint. The overall communication bus synthesis and supply voltage scaling under data variation is a nonlinear optimization problem, which is solved using a convex optimization tool. After applying our formulation to the optimization tool, it synthesizes the optimal bus width and the number of buses for the worst case scenario. Furthermore, it also finds the corresponding supply voltage for each communication task under variable workload, thus, it aims for the minimum energy consumption and the maximum bus utilization. In contrast to the voltage scaling model presented in **Chap.** 5, where almost all slack was exploited to scale voltage, in this chapter we control voltage scaling by introducing a new constraint so-called timing yield constraint, which finds a trade-off between energy consumption and communication cost. The experimental results show that a significant reduction of communication energy with an increasing timing yield constraint. However, it (timing yield constraint) offers a limitation to minimize the bus width and the number of buses, if the yield is increased beyond a certain limit.

As the second major contribution, we propose an extended model for the simultaneous communication bus synthesis and voltage scaling under data and process variations. According to the 2005 international technology roadmap for semiconductors (ITRS'05) [10] survey, dealing with fluctuations and statistical process variation in sub-

15nm CMOS technology will be a challenging task and there are quite a lot of questions left open for the designers. Recently, in [115, 40, 147] statistical timing analysis of a circuit under process variation was proposed. They showed that the adaptive body biasing technique can be used to mitigate the effects of process variations on the post-silicon circuit. However, non of the above approaches takes into account the problem of process variation during the synthesis of on-chip communication buses. We integrate the effect of data and process variations in our communication bus synthesis model and show that process variations have impacts on the synthesized communication bus in terms of communication cost. Thus, this again results in a trade-off between communication cost and energy consumption.

This chapter is organized as follows. **Sec.** 6.1 presents some preliminaries including motivational example and problem formulations for the simultaneous bus synthesis and voltage scaling under variations. **Sec.** 6.2 presents a formulation for combined bus synthesis and voltage scaling under data variation, which describes the modeling method for communication tasks, an optimization method, and supply voltage parameter estimation. **Sec.** 6.3 presents an extended model for both data and process variations. There, a delay model for communication tasks, an optimization algorithm, and parameter estimation of voltages are presented. Finally, **Sec.** 6.4 gives a summary of this chapter.

6.1 Preliminaries

We refer to the same kind of embedded systems as in **Chap.** 4 and 5, which are realized as an MPSoC that consists of several on-chip processing modules like general-purpose processors, ASICs or FPGAs. These on-chip modules communicate with each other by transferring data through communication buses like shared buses or point-to-point connections. After hardware/software partitioning and mapping, a target system with mapped tasks is shown in Fig. 6.1(a). Since a complex system runs a diversity of applications within a single chip, the total workload offered on the embedded system is not uniform over time. A typical example would be data flow intensive applications, such as voice and image processing, which can have different data size traffic for different scenarios. This causes a randomness on the data size to be transferred among on-chip communicating modules. We capture the communication tasks and its randomness of data size by profiling an application at system level for different scenarios and model data size as a random variable $NB(\zeta)$ (number of bit) with a known probability distribution function. In Fig. 6.1, random data size is assigned to each communication task c. We further assume that on-chip modules and interfaces are capable to scale supply and body bias voltages according to the workload of a system.

Based on the mapped tasks, a directed acyclic extended graph $G_E(T, E)$ is obtained

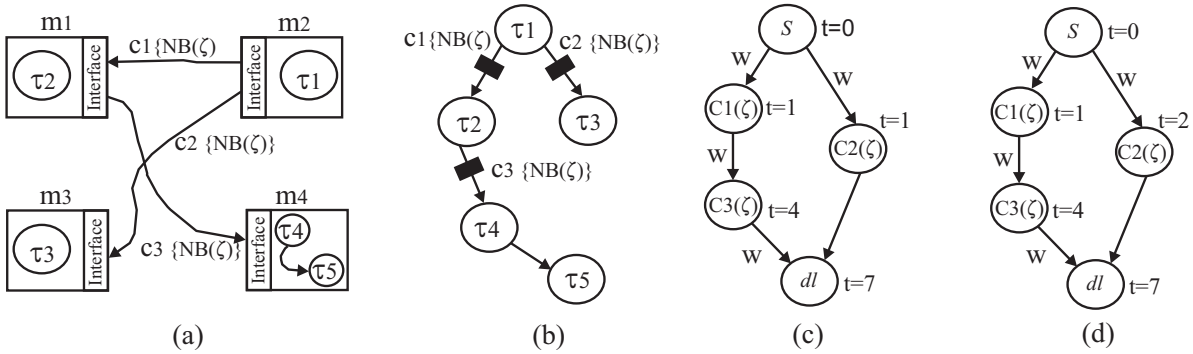


Fig. 6.1: Architecture and tasks model with variable data size. (a) Target architecture with mapped tasks and communication among them. (b) Extended tasks graph. (c) Communication task graph with ASAP scheduling of CLTIs for a 16-bit bus. (d) Communication task graph with ALAP scheduling of CLTIs for a 16-bit bus.

to extract the data processing tasks τ and the data communication tasks c of a given application. In the extended graph, a node $\tau \in T$ represents the data processing task, which is mapped onto an on-chip module, while edge $e \in E$ indicates data dependency. All communications over the on-chip communication buses are captured by communication task c_i along with its random data size to be transferred, as indicated by a square in Fig. 6.1(b). If the tasks τ_i and τ_j are mapped to the same module then there exist an edge between them without a square. This indicates that the tasks τ_i and τ_j do not communicate using an on-chip communication bus.

From the extended graph $G_E(T, E)$, a directed acyclic communication task graph $G_C(C, \Pi)$ is obtained with the start node *S* and deadline node *dl* to schedule the CLTIs of the communication tasks. In the communication task graph, a node $c \in C$ is a communication task, while an edge $\pi \in \Pi$ gives the dependency between the communication tasks. Fig. 6.1(c) depicts the communication task graph with ASAP scheduling of CLTIs for a 16-bit bus with the worst case scenario data size $3\sigma_{NB}$. An edge between two nodes c_i and c_j weighted with w is the data processing time of a task τ_i, which gives an early start time constraint for a successor c_j to transfer data using a communication bus. Fig. 6.1(d) depicts the ALAP scheduling of the CLTIs for a 16-bit bus for the worst case $3\sigma_{NB}$. In Fig. 6.1(c) and (d), there is a difference in ASAP and ALAP time for the node c_2. Since, the data size to be transferred among the on-chip modules is not deterministic, the slack is a random variable with a distribution similar to the data size. Their relation can be written as,

$$slack_{c,r,V_{dd},V_{bs}}(\zeta) = t_{ALAP_{c,r,V_{dd},V_{bs}}}(\zeta) - t_{ASAP_{c,r,V_{dd},V_{bs}}} \tag{6.1}$$

where $t_{ALAP_{c,r,V_{dd},V_{bs}}}(\zeta)$ can be expressed as,

$$t_{ALAP_{c,r,V_{dd},V_{bs}}}(\zeta) = dl_c - CLTI_{c,r,V_{dd},V_{bs}} \tag{6.2}$$

$$CLTI_{c,r,V_{dd},V_{bs}} = \left\lceil \frac{NB_c(\zeta)}{b_r} \right\rceil \cdot T_d \tag{6.3}$$

In above Eqs. (6.1), (6.2) and (6.3), dl_c is a deadline to finish a task, $CLTI_{c,r,V_{dd},V_{bs}}$ is the communication lifetime interval for a task c with a bus width r, supply voltage V_{dd}, and body bias voltage V_{bs}, $NB_c(\zeta)$ is a random data size to be transferred by a task, b_r is a bus of width r, and T_d is the time period of one clock cycle. For the sake of clarity, we consider only the supply voltage scaling for the dynamic energy consumption. Nonetheless, the leakage energy as well as the Adaptive Body Biasing (ABB) techniques [71, 107, 169, 17] can easily be incorporated into the formulation without changing our general approach. The alpha power delay model of a MOS transistor for one clock cycle with supply voltage V_{dd} can be written as,

$$T_d = \kappa \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{6.4}$$

where κ is a technology dependent constant, α is the saturation velocity ($1.4 < \alpha \leq 2$), V_{dd} is the supply voltage, and V_{th} is the threshold voltage. The dynamic energy consumption of each task is given by,

$$E_c = \alpha_\tau \cdot C_{eff} \cdot V_{dd}^2 \cdot T_d \tag{6.5}$$

where, α_τ is the switching activity of the communication tasks and C_{eff} is the effective switched capacitance for a data communication. The energy overhead, for switching from V_i to V_j, is

$$\varepsilon_{i,j}^{\Delta V} = C_r(V_i - V_j)^2 \tag{6.6}$$

where, C_r is the capacitance of the power rail. The time overhead, for switching from V_i to V_j, is given by

$$\delta_{i,j}^{\Delta V} = \rho|V_i - V_j| \tag{6.7}$$

where ρ is a constant.

6.1.1 Motivation

In **Chap.** 5, we gave a motivation for the simultaneous communication bus synthesis and voltage (supply and body bias) scaling for a system with a uniform workload, i.e., an on-chip data traffic is deterministic. In this subsection, we consider that the workload offered on an embedded system is not uniform so, we model the data size to be transferred between communication tasks c as a random variable with a known probability distribution function. We perform simultaneous scheduling, voltage scaling, bus selection, and binding of communication tasks c for random data sizes (for the worst case $3\sigma_{NB}$ of data size is considered) and illustrate how the data size can, in

the worst case scenario, influence the communication bus cost. Fig. 6.2(a) depicts CLTI delays as a function of two variables, voltage and the bus width for a fixed data size to be transferred between communication tasks. Fig. 6.2(b) shows a plot of CLTI delays for different scenarios with different amounts of transferred data over time and voltage constraints [T_{min}, T_{max}] and [V_{min}, V_{max}], respectively. Each scenario has a certain probability and they differ in terms of data size to be transferred. If we synthesize a communication bus considering scenario 1 with small amounts of transferred data and use the same communication bus for scenario 4 then the communication bus does not meet the given real-time constraints. In Fig. 6.2(b), it can be observed that scenario 1 meets the given time constraint [T_{min}, T_{max}] for voltages between 0.85V and 1.3V. While for scenario 4 with data size $3\sigma_{NB}$, the communication bus does not meet the time constraint. This motivates simultaneous scheduling, voltage scaling, bus selection, and binding of communication tasks considering a random data size.

Consider a system that has been partitioned and mapped onto the on-chip modules of an SoC and the driver of each module is capable to scale voltage while transferring data from one module to another. Furthermore, due to the diversity of applications to be run on a single embedded system, a task c can have variable data size $NB_c(\zeta)$ to be transferred. As shown in Fig. 6.1(a) first, module m_2 executes task τ_1 and its driver transfers the data to m_1 and m_3 in order to execute tasks τ_2 and τ_3, respectively. After receiving the data from module m_2, module m_1 executes task τ_2 and its driver transfers data to module m_4, which executes tasks τ_4 and τ_5. Task τ_5 has to be finished before the deadline of 7ms. The mean μ_{NB} and $3\sigma_{NB}$ of communication tasks c_1, c_2, and c_3 are 64 and 128-bit, respectively. The ASAP and ALAP scheduling of the communication task graph for μ_{NB} = 64-bit with start node and deadline node are shown in Fig. 6.1(c) and (d), respectively. Fig. 6.3(a) shows a scheduling with ASAP and ALAP time of tasks c_1, c_2 and c_3 for μ_{NB} = 64-bit, b_r = 32-bit and the nominal voltage settings (the highest supply voltage = 1.8V and body bias voltage = 0V), i.e., all drivers run at their maximum performance. This schedule of communication tasks c for a 32-bit bus results in the total slack (denoted by the white rectangle) of 4ms and needs a single bus to meet the time constraint of 7ms. From the given power consumption at the nominal voltage as shown in Fig. 6.3(a), the total energy consumption of all communication tasks can be calculated as 3·87mW = 261μJ. Fig. 6.3(b) shows a scheduling of the same communication tasks c_1, c_2, and c_3 for the worst case scenario with data size $3\sigma_{NB}$, b_r = a 32-bit bus and the nominal voltage. This schedule gives the total slack of 1ms and needs two separate 32-bit buses to meet the given deadline of 7ms. The total energy consumption at the nominal voltage can be calculated as, 2·174mW·3ms = 1044μJ. In Fig. 6.3(a) and (b), we saw that the synthesized single 32-bit bus for an average amount of data does not meet the real-time constraint for the worst scenario with size of data $3\sigma_{NB}$. Hence, two 32-bit buses are selected instead of a single 32-bit bus, however, the buses are underutilized when the workload offered to the system is low.

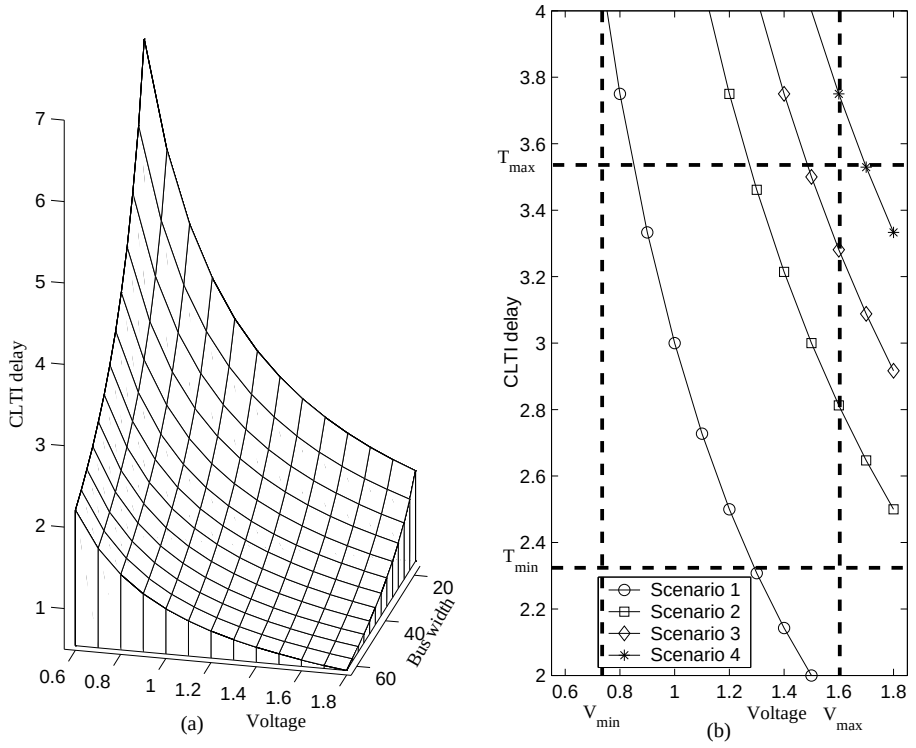


Fig. 6.2: Delay as a function of bus width and voltage. (a) CLTI as a function of bus width and voltage for a fixed data size. (b) CLTIs for variable data size for different scenarios

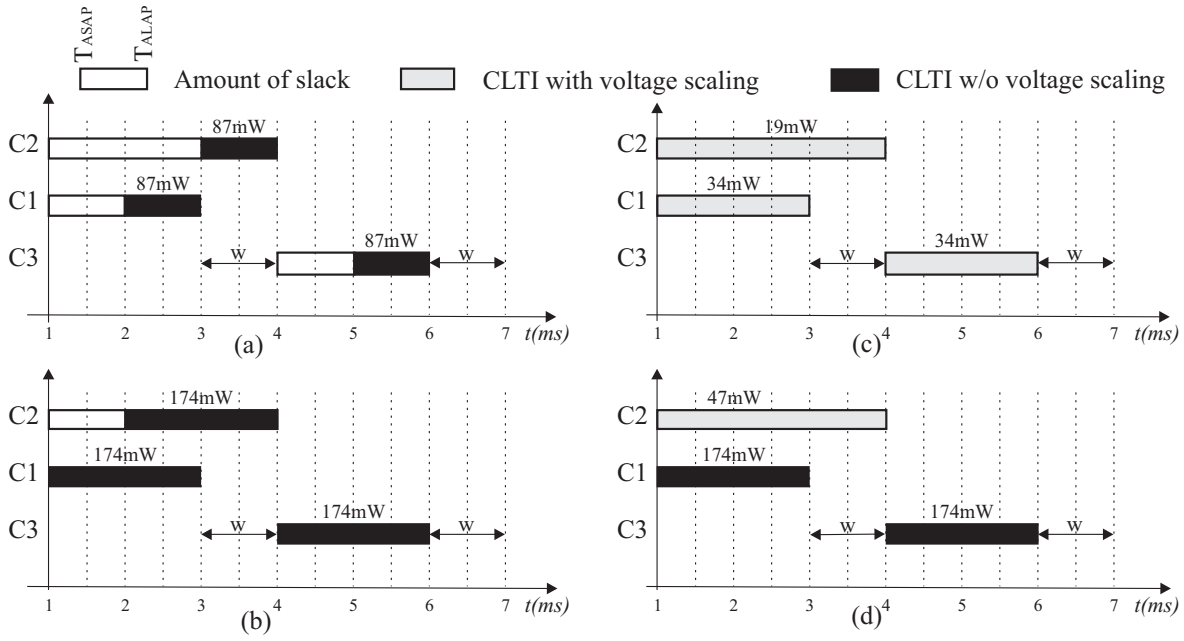


Fig. 6.3: Scheduling and voltage scaling of CLTIs for 32-bit bus. (a) Scheduling of CLTIs for deterministic data size. (b) Scheduling of CLTIs for $3\sigma_{NB}$ of random variable. (c) Scheduling and voltage scaling of CLTIs for deterministic data size. (d) Scheduling and voltage scaling of CLTIs for $3\sigma_{NB}$ of random variable.

In order to reduce the energy consumption of communication buses, the voltage is scaled to exploit the slack of each communication task and this results in a bus utilization factor constant over time as shown in Fig. 6.3(c) and (d). To make the problem simple, we assume in this example that the data processing time of each on-chip module is known to us, i.e, the operating voltage of modules are known. Furthermore, we assume that the supply voltage of all drivers can be varied continuously in the range [0.6, 1.8]V. In Fig. 6.3(c), communication tasks c_2 is scheduled with the supply voltage 0.9V to exploit a slack of 2ms, while tasks c_1 and c_3 are scheduled with voltage 1.4V to exploit a slack of 1ms. The total energy consumption of all tasks is 19mW·3ms+2·34mW·2ms = 193μJ, which corresponds to reduction in energy by 26$\%$ compared to the energy consumption at the nominal voltages of two 32-bit buses. In Fig. 5.2(d) depicts scheduling and voltage scaling of communication tasks shown in Fig. 5.2(b) under data size $3\sigma_{NB}$ = 128-bit for all tasks. In Fig. 5.2(d) task c_2 has a slack of 1ms and this slack is exploited by scaling the supply voltage to 1.2V, while tasks c_1 and c_3 do not have any slack and their voltages are kept to the nominal voltage so that communication takes place within two 32-bit synthesized buses. The total energy consumption is calculated as 47mW·3ms+ 2·174mW·2 = 398μJ, which is a reduction in energy consumption by 19$\%$ compared to the scheduling of tasks at nominal voltage as shown in Fig. 6.3(b).

6.1.2 Problem Formulation

As in previous chapters, we assume that a set of tasks have been partitioned and mapped efficiently onto the appropriate modules of an SoC. Each module m_i has data processing tasks τ and a communication task c that transfers data to another module m_j, which has a data dependency. The data transfer from one module to another module takes place via a communication bus and its interfaces and they are driven by drivers that are capable to scale the voltage during each data transfer. An interface attached to each on-chip module establishes communication such as bus requests, data transfer, and the release of the bus after successful completion of data transfers. Due to the diversity of applications that can run on a single SoC, the workload offered to the embedded system is not uniform over the time. This introduces a certain randomness on the data size to be transferred among the on-chip communication tasks. We model the amount data to be transferred by a communication task c as a random variable $NB_c(\zeta)$ with a known probability distribution function. For each task c its ASAP time, ALAP time, deadline dl_c, the distribution of random variable $NB_c(\zeta)$ and the switched capacitance C_{eff} are given. Based on the mapped tasks τ, a directed acyclic extended graph $G_E(T, E)$ is obtained as shown in Fig. 5.1. From the extended graph $G_E(T, E)$, the communication task graph $G_C(C, \Pi)$ is obtained with start node S and deadline node dl. In the communication task graph $G_C(C, \Pi)$, $c \in C$ be a set of commu-

nicating tasks and their data dependency between the communication tasks is defined by a set $\Pi \subseteq (C \times C)$, consists of two-tuples (c_i, c_j) where a successor c_j depends on the results of the predecessor c_i. This data dependency between communication tasks is constrained by a set $Depn \subseteq (C \times C \times W)$ consists of 3-tuples (c_i, c_j, w) such that $\forall i, j \in [1 \ldots N]$, $(c_i, c_j)_{i \neq j} \in \Pi | \Pi \subseteq C \times C$, a task c_j can start transferring data no earlier than w time units after the completion of transferring data by c_i.

Furthermore, we assume that the supply voltage V_{dd} and the body bias voltage V_{bs} of each data processing task τ_i are known and its corresponding execution time can be calculated using Eqs. (4.1) and (4.2). Unlike this, the supply voltage V_{dd} and the body bias voltage V_{bs} of each communication task $c \in C$ are unknown and to be identified. In this work for the sake of clarity, we consider only the supply voltage scaling, however, adaptive body biasing (ABB) can easily be incorporated in our approach of communication bus synthesis. Each task $c \in C$ can vary its supply voltage V_{dd} within a certain continuous and discrete voltage range. At the same time, the body bias voltage V_{bs} is kept to 0V.

6.2 Combined Bus Synthesis and Voltage Scaling Under Data Variation

As on-chip data traffic is not uniform over time, communication bus synthesis and voltage scaling techniques discussed in previous **Chap.** 4 and 5 do not find the best solution. In this section we present a method to model communication tasks c as a function of the random variable $NB_c(\zeta)$ and cast that model to a deterministic constraint, which can be solved efficiently using any nonlinear convex optimization tool. After solving the problem using an optimization tool, the bus width, the number of buses, and the supply voltage for each communication task will be obtained for the worst case scenario. However, later in **Sec.** 6.2.3 we estimate the probability distribution function of voltage for a given synthesized bus width with a variable workload. It shows how voltage is distributed over a range of different data size to be transferred.

6.2.1 Modeling of Communication Tasks

The relation of the data transfer delay CLTI with the bus width b_r, supply voltage V_{dd}, body bias voltage V_{bs}, and a random data size $NB_c(\zeta)$ of a communication task c are given by Eqs. (6.3) and (6.4). In this subsection we consider, as commonly assumed in the literature [120, 46], that the CLTI is inversely proportional to V_{dd} (V_{th} = 0, $\alpha = 2$) to make the illustration of our point simpler, however, the drawn conclusions are valid for the general case. After the simple algebraic manipulation of Eqs. (6.3) and (6.4) we

get,

$$CLTI_{c,r,V} = \kappa \frac{NB_c(\zeta)}{b_r \cdot V_{dd}} \tag{6.8}$$

where the data transfer delay CLTI is a function of a random variable $NB_c(\zeta)$ such that the delay itself is also a random variable scaled by constant parameters κ, bus width b_r, and supply voltage V_{dd}. Let a term η be the timing yield constraint of a communication task c so that a probabilistic delay constraint of each communication task can be formulated as,

$$\forall c \in C,\ P(dl_c - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}} \geq 0) \geq \eta \tag{6.9}$$

where, dl_c is the deadline of a communication task c and $\delta_{i,j}^{\Delta V}$ is the delay overhead due to supply voltage from one level to another level switching. Eq. (6.9) denotes the probability that the sum of a random variable $CLTI_{c,r,V_{dd}}$ and the deterministic switching overhead delay $\delta_{i,j}^{\Delta V_{dd}}$ is less than or equal to the deadline dl_c for each communication task c. The timing yield constraint η can be considered to be a confidence level, i.e., the probability of each task c having a delay less than its deadline to be more than or equal to η. We assume that the data model of all communication tasks c are normally distributed random variables with the mean μ_{NB}, and the standard deviation σ_{NB}. However, the proposed formulation can be used for any arbitrary distribution. Combining Eqs. (6.8) and (6.9), the probabilistic constraint of each communication task c can be formulated in terms of mean and standard deviation of CLTI as,

$$\begin{aligned} &\forall c \in C,\ P((dl_c - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V}) \sim \\ &\mathcal{N}(\mu_{CLTI}(NB), \sigma_{CLTI}(NB))) \geq 0) \geq \eta \end{aligned} \tag{6.10}$$

which can be rewritten as

$$\forall c \in C,\ dl_c - \mu_{CLTI}(NB) - \ \phi^{-1}(1-\eta)\ \sigma_{CLTI}(NB) \geq 0 \tag{6.11}$$

where $\phi^{-1}(\cdot)$ is the inverse of the error function. We assume that the timing yield is constrained by the range $0.5 < \eta \leq 1$ so, Eq. (6.11) can be considered as a convex function under the condition that $\eta > 0.5$. Since the target yield for a given path is always much greater than 50%, this condition is easily satisfied. The proof of convexity of Eq. (6.11) is given in Lemma 6.2.1.

Lemma 6.2.1 *The nonlinear delay constraint given in Eq. (6.11) is a convex function under the condition that $\eta > 0.5$.*

Proof Note that a nonnegative weighted sum of convex functions is also a convex function [86]. The mean $\mu_{CLTI}(NB)$ in Eq. (6.11) can be written in the form,

$$\mu_{CLTI}(NB) = \frac{1}{n} \sum_{c} \frac{NB_c(\zeta)}{b_r \cdot V} \tag{6.12}$$

where, $\mu_{CLTI}(NB)$ is an arithmetic average of the random samples and that is an inverse function of the optimization variable b_r (bus width). By applying the Jensen's inequality [86] over the term $\mu_{CLTI}(NB)$,

$$\frac{NB_c(\zeta)}{b_r \cdot V} \leq \lambda \cdot \frac{NB_c(\zeta)}{b_{r_1} \cdot V} + (1-\lambda) \cdot \frac{NB_c(\zeta)}{b_{r_2} \cdot V} \tag{6.13}$$

for $0 \leq \lambda \leq 1$ the above inequality exists. Hence, the mean $\mu_{CLTI}(NB)$ is a convex function. To proof the convexity in $\sigma_{CLTI}(NB)$, let us first simplify the expression of a standard deviation,

$$\begin{aligned} g(x) &= \frac{1}{n-1}\sum_{c}(X_c - \mu)^2 \\ &= \frac{1}{n-1}(X_c - \mu)^T(X_c - \mu) \end{aligned} \tag{6.14}$$

where, $X_c = (x_1, x_2, \cdots x_n)$ is a vector of point masses and $\mu = (\mu_1, \mu_2, \cdots \mu_n)$ is also a vector of mean. After a simple algebraic manipulation of Eq. (6.14), the above term can be written as follows,

$$\frac{1}{n-1}(2I - X_c^T\mu - \mu^T X_c) \tag{6.15}$$

where, X_c, μ are both vectors and I is an identity matrix. We evaluate the Hessian matrix $H(x) = \nabla^2 g(x)$ by taking the second derivative of the Eq. (6.15) with respect to x then we get,

$$\begin{aligned} &\frac{1}{n-1}\frac{\partial}{\partial x}(2I - X_c^T\mu - \mu^T X_c) \\ &\frac{1}{n-1}\frac{\partial}{\partial x}(2I - 4e_c^T e_c) = 0 \end{aligned} \tag{6.16}$$

where, $e_c = (1, 1, \cdots 1)$ is a vector with elements equal to one. The Hessian matrix $\nabla^2 g(x) = 0$ is positively semidefinite and hence, Eq. (6.15) is a convex, i.e. the standard deviation $\sigma_{CLTI}(NB)$ is a convex function.

Since, $\mu_{CLTI}(NB)$ and $\sigma_{CLTI}(NB)$ are both convex, the left hand side of the constraint Eq. (6.11) is convex iff [25]

$$\phi^{-1}(1-\eta) \leq 0 \tag{6.17}$$

which is exactly the case iff $\eta > 0.5$.

6.2.2 Optimization Methodology

The on-chip communication bus synthesis and voltage scaling problem under uncertainty of data size is a multi-variable nonlinear optimization problem with a probabilistic delay constraint. This problem is relaxed to the convex quadratic optimization

problem and can be solved efficiently using a convex optimization tool. For the convex optimization problems an exact solution can be found as a local optimal solution is also the global optimal solution.

Problem 6.2.2.1 *(Combined scheduling, supply voltage scaling, bus selection, and binding of communication task $c \in C$ to minimize the communication bus width and number of buses with reduced communication energy under the variation of on-chip data traffic) Perform simultaneous scheduling, supply voltage scaling, bus selection, and binding of communication task $c \in C$ that minimize the communication cost shown in Eq. (5.11), subject to: $\forall c \in C$, $P((dl_c - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}}) \geq 0) \geq \eta$ and $V_{dd_{min}} \leq V_{dd} \leq V_{dd_{max}}$ for all $t \in \{0 \cdots \lambda\}$. The first constraint is a probabilistic delay constraint of communication task c such that the overall delay of a task c should be less than or equal to its given deadline dl_c with the probability η, which is called timing yield constraint. While, the second constraint gives the limit for supply voltage scaling such that they cannot be scaled beyond their limits.*

The formulation of the simultaneous on-chip communication bus synthesis and supply voltage scaling problem with nonlinear probabilistic delay constraint is very close to the formulation presented in **Sec.** 5.3. The objective function is to minimize total communication bus cost (i.e., the bus width and the number of buses), which is given in Eq. (5.11). The binding constraint for each communication task c to a communication bus with width r and supply voltage V_{dd} is given in Eq. (5.12), where the binary decision variable $X_{c,t,r,V_{dd}} \in \{0, 1\}$ indicates that the scheduling of communication task c, at time $t \in \{0 \cdots, \lambda\}$ with bus b of width r and supply voltage V_{dd}. At time t a communication bus b of width r cannot be used by more than one communication tasks $(c, c^{'}) \in C$, if their CLTIs overlaps with each other then two separate buses are assigned to them. This constraint is given in Eq. 5.15). Similarly, the dependency constraint between communication tasks c and the continuous supply voltage scaling constraint are given in Eqs. (5.16) and (5.17), respectively. The probabilistic delay constraint of each communication task c with timing yield constraint η is given in Eq. (6.18).

$$\forall\, c \in C,\ P\left(\left(\sum_{r \in R} \sum_{\substack{t= \\ ASAP_c}}^{\Psi} (dl_c - t - CLTI_{c,r,V_{dd}} - \delta_{i,j}^{\Delta V_{dd}}) \cdot X_{c,t,r,V_{dd}}\right) \geq 0\right) \geq \eta \qquad (6.18)$$

Where, dl_c is the deadline of each communication task, $CLTI_{c,r,V_{dd}}$ is the communication lifetime interval of task c, with bus width r and supply voltage V_{dd} and $\delta_{i,j}^{\Delta V_{dd}}$ is the overhead delay due to switching of supply voltage. The constraint indicates that the overall delay of communication task c should be less than or equal to deadline dl_c with the probability η, which defines a confidence level.

CONVEXOPTALGORITHM()
1 Find the center x_c of the current polytope P.
2 If $x_c \notin S$, find the gradient $\nabla g_c(x)$ of the constraint having the largest value at x_c.
3 Insert a hyperplane of the form $c^T x \geq \beta = c^T x_c$, where $c = -[\nabla g_c(x)]^T$.
4 Update P.
5 If $x_c \in S$, compute $c = -[\nabla f(x)]^T$.
6 Insert the hyperplane of the form $c^T x \geq \beta = c^T x_c$. update P.
7 If the size of the polytope P is less than a user specified limit ϵ, stop.
8 Otherwise goto Step 1.

Algorithm 6.1: Convex optimization algorithm.

We cast the above simultaneous on-chip communication bus synthesis and voltage scaling problem with nonlinear probabilistic constraints to a convex quadratic optimization problem. This problem can be solved efficiently using the interior point methods [5] in a *quasi polynomial* time complexity. The corresponding proof is given in **Sec.** 5.1.3. Since the convex quadratic constraint is a convex function for $\eta > 50\%$, it guarantees a globally optimal solution as proved in Lemma 6.2.1. The continuous voltage scaling of communication tasks c during each data transfer gives the minimum possible total energy consumption of a communication bus, however, due to the practical reasons continuous voltage scaling technique is not considered. Therefore, the discrete voltage scaling heuristic presented in **Sec.** 5.3.2 finds a near-optimal solution in a polynomial time complexity.

6.2.2.1 Optimization Algorithm

The algorithm works by successively reducing the problem region by introducing cutting planes in every iteration. The cutting planes (or hyperplanes) are obtained by conditions on the gradient of the objective functions and that of the constraints as shown at line 2. The cutting planes are chosen such that they guarantee the presence of the optimal solution in the problem region of the next iteration. Let $x \in \mathbb{R}^{+n}$ be a decision variable, $f(x)$ be the convex objective function and $g_c(x) \leq dl_c$, *i = 1,· · ·,n* be the convex constraints. Let *S* be the feasible set defined by $\{x : g_c(x) \leq dl_c\}$ and $x^* \in S$ be the optimal solution. Initially, the solution space is determined by the polytopes defined by the set $\{x : x_L \leq x \leq x_U\}$, where x_L and x_U are the minimum and maximum possible values of x.

6.2.2.2 Timing Yield Search Algorithm

Algorithm 6.2 is a search algorithm to find the best timing yield constraint η, with reduced energy and the minimum communication cost. At the start of the algorithm, *CurrYield* is set to *MaxYield* and each iteration it is decremented. In the algorithm, line 1-4 is for the initialization, where *MaxYield*, *MinYield*, and *MinStep* are set to 99%, 51%, and 1%, respectively. The term *step* is a variable to decrease the timing yield constraint in each iteration. At line 6 to 14, communication cost is computed for different values of the timing yield constraint and in each iteration *CurrYield* is decremented by *step*. If the condition given at line 15 satisfies, the algorithm leaves the while loop. At line 15, if the current communication cost (*CurrCommCost*) with a higher value of timing yield constraint is not equal to the previous communication cost (*PrevCommCost*) with lower value of timing yield constraint, the condition satisfies. At line 17, we check the difference in current and previous timing yield; if it is greater than 0.1 then *BusTunning* function is called recursively, else, the algorithm returns the best timing yield constraint (*BestYield*), with optimal communication cost and communication energy consumption.

The run time of the algorithm depends on *MaxYield*, *MinYield*, and *MinStep*. For the worst case, the number of iterations in the while loop can be obtained as (*MaxYield-MinYield*)/*MinStep*.

6.2.3 Parameters Estimation of Voltage

In this section, we present a model to estimate the probability distribution function of supply voltage V_{dd} under a random data size to be transferred among the on-chip modules. When the random data model of communication tasks c presented in **Sec.** 6.2.1, is applied to the nonlinear optimization algorithm, the optimal bus width $b_r(opt)$ for each communication task can be obtained. After an algebraic manipulation of Eqs. (6.3) and (6.4), the statistical parameters of supply voltage in terms of the optimal bus width $b_r(opt)$ can be formulated as,

$$V_{dd} = \kappa \frac{NB_c(\zeta)}{b_r(opt) \cdot CLTI_{c,r_{opt},V_{dd}}} \tag{6.19}$$

In Eq. (6.19) the data transfer delay $CLTI_{c,r_{opt},V_{dd}}$ is a variable, which is a function of the synthesized bus width $b_r(opt)$ and variable supply voltage V_{dd}. The main goal is to exploit the slack for the reduction of total energy consumption of the communication bus such that the delay $CLTI_{c,r_{opt},V_{dd}}$ will be closed to the deadline dl_c of a communication task c. Hence, in Eq. (6.19), the delay $CLTI_{c,r_{opt},V_{dd}}$ should be replaced by a delay, which is a function of the timing yield constraint η. Above equation can be rewritten

```
TIMINGYIELDTUNNING()
 1  MaxYield ← GETMAXYIELD();
 2  MinYield ← GETMAXYIELD();
 3  MinStep ← GETMAXYIELD();
 4  γ ← GETGAMMA();
 5  step = (MaxYield − MinYield)/γ;
 6  /* computation of communication cost for different timing yield */
 7  int BUSTUNNING(MaxYield, MinYield, step){
 8      CurrYield ← MaxYield;
 9      PrevYield ← MaxYield;
10  do {
11      CurrCommCost ← COMPUTEBUSCOST(CurrYield);
12      PrevCommCost ← COMPUTEBUSCOST(PrevYield);
13      PrevYield ← CurrYield;
14      CurrYield ← CurrYield − step;
15      }while (CurrCommCost == PrevCommCost);
16  /* check for the best timing yield constraint*/
17  if (CurrYield − PrevYield)/100 > 0.1
18    then
19          step ← MinStep;
20          BUSTUNNING(CurrYield, PrevYield, MinStep);
21          else
22          BestYield ← PrevYield;
23
24  return BestYield;
```

Algorithm 6.2: Algorithm to search for the best timing yield constraint.

as,

$$V_{dd} = \mathcal{K}_{V_{dd}} \cdot NB_c(\zeta) \tag{6.20}$$

$$\mathcal{K}_{V_{dd}} = \frac{\kappa}{b_r(opt) \cdot dl_c \cdot \mathcal{P}_{V_{dd}}} \tag{6.21}$$

$$\mathcal{P}_{V_{dd}} = \frac{\mu_{CLTI} + \phi^{-1}(1-\eta) \cdot \sigma_{CLTI}}{dl_c - [\mu_{CLTI} + \phi^{-1}(1-\eta) \cdot \sigma_{CLTI}]} \tag{6.22}$$

In Eq. (6.21) the mean μ_{CLTI} is a constant, while the standard deviation σ_{CLTI} changes for different values of timing yield constraint η. However, for a given timing yield constraint η the optimization algorithm presented in **Sec.** 6.2.2 gives a fixed delay with an achieved yield for supply voltage scaling. Hence, the probability density function of supply voltage under variable data traffic can be written as [125, 136],

$$f_{V_{dd}}(nb) = \left(\frac{1}{\mathcal{K}_{V_{dd}}}\right) f_{NB}\left(\frac{nb}{\mathcal{K}_{V_{dd}}}\right) \tag{6.23}$$

In communication task graph $G_c(C, \Pi)$ as shown in Fig. 6.1, random variables (data size to be transferred among communication tasks) are independent and each communication task can have a different distribution function of random data size such that the resulting distribution of supply voltage is not identical for all communication tasks c. The overall distribution of supply voltage including all communication tasks can be written as a sum of individual distributions,

$$f_{V_{dd}} = \sum_{i=1}^{|C|} f_{V_{dd_i}} \tag{6.24}$$

In Eq. (6.24) the sum of individual supply voltage distributions can be computed by convolving individual distributions,

$$f_{V_{dd}} = f_{V_{dd_i}} * f_{V_{dd_{i+1}}} * \cdots * f_{V_{dd_C}} \tag{6.25}$$

Similarly, the mean slack $\hat{\mu}_{Slack}$ can be estimated as,

$$\begin{aligned} \hat{\mu}_{slack} &= E[dl_c - \kappa \frac{NB_c(\zeta)}{b_r(opt) \cdot V_{dd}} - t_{ASAP_c}] \\ &= dl_c - \frac{\kappa}{b_r(opt) \cdot V_{dd}} E[NB_c(\zeta)] - t_{ASAP_c} \end{aligned} \tag{6.26}$$

6.3 Extension to Process Variation

In **Sec.** 6.2 we presented a model to synthesize the optimal bus width and the number of buses under data size variation of communication tasks. The gate delay model was

based on the alpha-power delay model [29] that is a deterministic model without taking into account variations. In this section, we present an extended model that takes into account both data size variation and effect of process variations on the short channel devices. As a result of this, the synthesized on-chip bus is robust against variable on-chip data traffic and process variations.

The effect of process variations is increasing severely on the deep sub micron technology as feature sizes continue toward the sub-100nm era. As a result of this, a synthesized digital circuit can have a completely different performance than the expected. We address this problem by proposing a process variations aware simultaneous on-chip communication bus synthesis and voltage scaling technique in presence of random data size to be transferred among on-chip communicating modules. The method finds an energy efficient the optimal bus width and the number of buses by mitigating the effect of process variations. The slack is exploited to maximize bus sharing and to reduce energy consumption by simultaneously scaling the voltages (supply and body bias) during the synthesis of on-chip communication buses. The resulting synthesis problem is relaxed to the convex quadratic optimization problem and is solved efficiently using a convex optimization tool. The effectiveness of our approach is demonstrated by applying optimization to an automatically generated benchmark and a real-life application. By varying the value of timing yield constraint, a trade-off between minimization of communication bus cost and energy consumption is explored in presence of process variations. The experimental results show a significant reduction in communication energy with the increasing timing yield constraint. However, the timing yield constraint offers a limitation to minimize the communication cost and the effect of process variations, if the yield is increased beyond a certain limit. i.e., at a high amount of slack the effect of process variations is negligible on the communication bus cost and the energy consumption. While, at a low amount of slack the effect of process variations is significant and results in an increased in communication bus cost and energy consumption to meet the real-time constraints. Furthermore, we also estimate the probability distribution function of the voltages in presence of random data size and process variations.

6.3.1 Overview and Contributions

The process variations encompass several variation parameters including channel length L, threshold voltage V_{th}, thickness of oxide T_{ox} and channel width W, which are due to manufacturing phenomena. In [26, 52, 84] worst case analysis and optimization of VLSI circuits performance are presented under process variations. Worst case analysis refers to the process of determining the values of the noise parameters in the worst case conditions and the corresponding worst case circuit performance values. How-

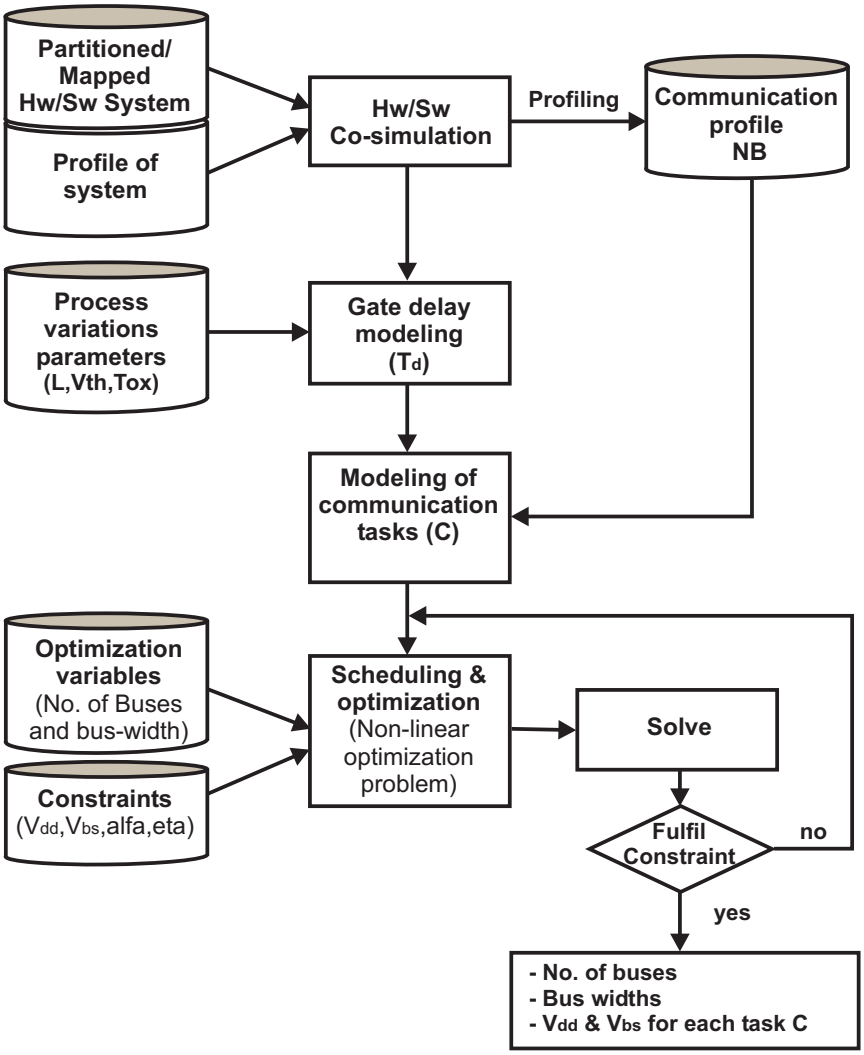


Fig. 6.4: Design flow for on-chip communication bus synthesis and voltage scaling under data size and process variations

ever, the worst case analysis approaches are too pessimistic and lead to extremely conservative designs, which cause a bottlenecks due to inaccurate worst case models. In [135, 146, 106] probabilistic frameworks are proposed to model and analyze the effects of process variations. These approaches give a better solution than the conventional worst case analysis methods.

We use the voltage scaling technique to mitigate the effect of process variations and to reduce the energy consumption of the on-chip communication buses. Fig. 6.4 depicts the design flow of a variation aware on-chip communication bus synthesis and voltage scaling technique. The method takes a partitioned and mapped hardware/software system with their system level profile and performs co-simulation to trace the communication events and statistics of amount of data to be transferred between on-chip modules. From the given process variation parameters L, V_{th}, T_{ox} and W, a statistical gate

delay T_d is estimated. All three informations, which are statistical gate delay T_d, communication events, and their statistics are combined in order to obtain the statistical model of communication tasks. The variation aware on-chip communication bus synthesis and voltage scaling problem is later casted into a scheduling and optimization problems with their optimization variables and constraints. The optimization problem is a probabilistic nonlinear optimization problem, which can be solved efficiently using a nonlinear quadratic optimization tool. The optimization tool finds the optimal bus width, the number of buses, and assigns voltages (supply and body bias) for an individual communication task.

6.3.2 The Sources of Variations

First, as we have seen in **Chap.** 4 and 5 in the presence of diversity of applications to be run on a single embedded system, the workload offered to the embedded system is not uniform over time. A typical example could be in a partial reconfigurable platform, where some of the on-chip modules are reconfigured partially to meet a dynamic demand of workload. This introduces the randomness on the data size to be transferred among the on-chip communicating modules.

Second, the semiconductor manufacturing variation occurs when parameters deviate from their ideal or designed values. As technology scales, the importance of understanding variation is increasing further. The variation in performance of integrated circuits can be categorized into temporal and spatial sources [115]. The temporal sources vary over time and depend on circuit operating conditions. Example for these include effects such as switching activity and temperature variation. The spatial effects are fixed in time and depend on physical factors such as structural variation in the chip that is based on the circuit layout, neighboring environment, and process conditions. The spatial variation sources impact the geometry of a structure and can lead to undesirable effects such as yield loss. The most important sources of device variation are L, T_{ox}, V_{th} (threshold voltage) and W. Fig. 6.5 depicts the general trend in the ratio between within-die and total variation for some key technology device and wire parameters [115]. We can see that the within-die proportion of L variation increases from 40% to 65%. The variation of the wire geometry parameters, width W, height H, thickness T_{ox}, and resitivity ρ also quite big. Other parameter variations such as the oxide thinkness T_{ox} and threshold voltage V_{th} increase at a lower rate as shown in Tab. 6.1. Models and methods for dealing with such variation trends will become an increasingly important part of high performance circuit design.

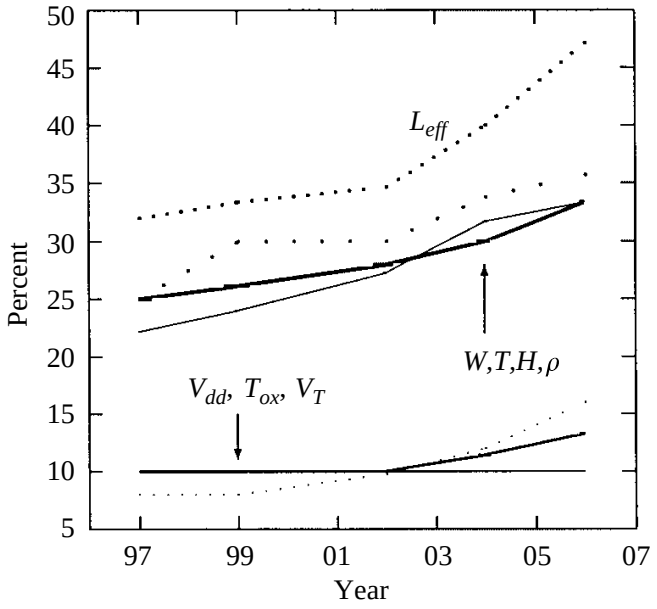


Fig. 6.5: Variation in device and interconnect [115]

Year	Channel length L_{eff}(nm)	Oxide thickness T_{ox}(nm)	Threshold volt. V_{th}(mV)	Channel width W(μm)	Channel height H(μ m)
1997	250±80	5.0±0.40	500±50	0.80±0.20	1.2±0.3
1999	180±60	4.5±0.36	450±45	0.65±0.17	1.0±0.3
2001	130±45	4.0±0.39	400±40	0.50±0.14	0.9±0.27
2003	100±40	3.5±0.42	350±40	0.40±0.12	0.8±0.27
2005	70±33	3.0±0.48	300±40	0.30±0.10	0.7±0.25

Tab. 6.1: Technology parameters and their 3σ variations [115]

6.3.3 Delay Model

In this subsection, we present delay models for gates and communication tasks separately, as a function of technological parameters and data size variations. Since the α-power delay model used in **Chap.** 4 and 5, does not model accurately the effect of short channel effects of CMOS devices, we present a model that explicitly models these effects and process variations. The resulting delay model of a gate is expressed in terms of variation parameters such as channel length, threshold voltage, and velocity saturation effect. Based on the sensitivity analysis method, we estimate the mean delay and the delay variance. Later, these estimated delay parameters are used to model the delay of communication tasks.

6.3.3.1 Gate Delay Model

The main goal is to model the impact of gate delay variations due to within-die process variations on the circuit delay. Thus, the statistical delay model of a gate as a function of process variation parameters such as channel length, threshold voltage, and velocity saturation effect can be written as,

$$T_d(\zeta) = \frac{K(\zeta) \cdot V_{dd}\{1 + \frac{V_{dd}-V_{th}(\zeta)}{E_{sat} \cdot L(\zeta)}\}}{\{\frac{V_{dd}-V_{th}(\zeta)}{2S(\zeta)}\}^2} \tag{6.27}$$

where E_{sat} is the electric field for the velocity saturation. As the channel length L becomes shorter, threshold voltage V_{th} exhibits a greater dependence on L and drain bias. Larger supply voltage V_{dd} and smaller L usually lead to a sharp degradation of V_{th} (i.e., V_{th} roll-off) and thus, shorter gate delay. Accurate modeling of V_{th} as a function of L and V_{dd} is important for an accurate gate delay model. Based on the physical derivations in BSIM [121], we simplify the model of V_{th} roll-off as [36],

$$V_{th}(\zeta) = V_{th0} - K_1 \cdot V_{bs} - K_2 \cdot V_{dd} \cdot exp\,(-\alpha_{DIBL} \cdot L(\zeta)) \tag{6.28}$$

where V_{th0} is the long-channel threshold voltage V_{th}, K_1, and K_2 are fitting parameters and α_{DIBL} is the DIBL (Drain Induced Barrier Lowering) coefficient. Both values can be extracted from transistor characteristics. Note that for some technologies where heavy halo implantation is employed, V_{th} roll-up can also be apparent. In that case, another term in the order of $L^{-1/2}$ should be added to Eq. (6.28). Experimental data shows that the sub-threshold swing *(S)* is also a function of *L*, sharing similar exponential dependence as the DIBL effect. Up to the first order, it can be model as,

$$S(\zeta) = S_0 \cdot [1 + exp\,(-a_s \cdot L(\zeta))] \tag{6.29}$$

The parameter $K(\zeta)$ in Eq. (6.27) is a random variable, which is expressed as a polynomial function of L and the loading capacitance that represents the dependence of

gate delay on loading and is normalized to (W/L). Since the ratio of W/L determines the drain current of a CMOS transistor, if W is much larger than L, variation in W is usually not considered. Hence, the parameter W is considered to be a deterministic.

$$K(\zeta) = [k_0 + k_1 \cdot L(\zeta) \cdot C_{load} + k_2 \cdot L^{a_k}(\zeta)]/W \tag{6.30}$$

The gate delay T_d in Eq. (6.27) is a random variable. We assume that the magnitude of process variations parameters is small, hence the first order delay model of T_d as a function of variations can be written as,

$$T_d = \mu_{T_d} + \sum_{i=1}^{n} \frac{\partial T_d}{\partial x_i} \cdot \sigma_{x_i} + \frac{\partial T_d}{\partial x_{i+1}} \cdot \sigma_{x_{i+1}} + \frac{\partial T_d}{\partial x_i, x_{i+1}} \cdot \sigma_{x_i, x_{i+1}} \cdot \rho_{x_i, x_{i+1}} \tag{6.31}$$

where, μ_{T_d} is the mean of random gate delay T_d, which is estimated using Eq. (6.27) for mean values of process variation parameters x_i. The term σ_{x_i} is the standard deviation of a parameter x_i and $\rho_{x_i,x_{i+1}}$ is the cross correlation factor between parameters x_i and x_{i+1}. In this work, we assume that all variation parameters are independent random variables, hence the cross correlation between two random variables $\rho_{x_i,x_{i+1}}$ = 0. The first order delay model of T_d as a function of process variation parameters L, V_{th}, T_{ox} and W can be written as,

$$T_d = \mu_{T_d} + \frac{\partial T_d}{\partial L} \cdot \sigma_L + \frac{\partial T_d}{\partial V_{th}} \cdot \sigma_{V_{th}} + \frac{\partial T_d}{\partial T_{ox}} \cdot \sigma_{T_{ox}} \tag{6.32}$$

From Eq. (6.32) the variance of delay T_d can be obtained as,

$$\sigma^2_{T_d} = \left(\frac{\partial T_d}{\partial L}\right)^2 \cdot \sigma^2_L + \left(\frac{\partial T_d}{\partial V_{th}}\right)^2 \cdot \sigma^2_{V_{th}} + \left(\frac{\partial T_d}{\partial T_{ox}}\right)^2 \cdot \sigma^2_{T_{ox}} \tag{6.33}$$

Let, $T_{critical}$ be the critical delay of a path in a given integrated circuit. Under the process variations, the gate delay T_d of each communication task $c \in C$ should be less than or equal to $T_{critical}$. The probabilistic constraint can be expressed as,

$$\forall c \in C,\ P(T_d \leq T_{critical}) \geq \alpha \tag{6.34}$$

where, $P(\cdot)$ denotes the probability that the random variable T_d is less than or equal to $T_{critical}$ with a probability greater than or equal to α. The notation α can be considered to be a confidence level. We assume that the variation in gate delay under the process variations, has a normal distribution. Thus, the probabilistic constraint of Eq. (6.34) can be transferred into the deterministic constraint with a function of mean μ_{T_d} and standard deviation σ_{T_d} of a random variable T_d.

$$T_{critical} - \mu_{T_d} - \phi^{-1}(\alpha) \cdot \sigma_{T_d} \geq 0 \tag{6.35}$$

where, the term $\phi^{-1}(\cdot)$ is the inverse of an error function. In Eq. (6.35), it can be noticed that the acceptance or rejection of gate delay T_d under process variation depends on the chosen confidence level α. If the confidence level is high, the optimization algorithm selects the delay with low variation to meet the time constraint $T_{critical}$. In this work, we set the confidence level α for each task to 99%.

6.3.3.2 Delay Model of Communication Task

The data transfer delay CLTI for each communication task $c \in C$ as a function of bus width b_r, random data size $NB_c(\zeta)$, and a random gate delay $T_d(\zeta)$ can be written as,

$$CLTI_{c,r,V_{dd},V_{bs}} = \left\lceil \frac{NB_c(\zeta)}{b_r} \right\rceil \cdot T_d(\zeta) \tag{6.36}$$

In Eq. (6.36) the data transfer delay CLTI is a function of two random variables and its probability distribution function can be obtained as,

$$\begin{aligned} F_{CLTI_{c,r,V_{dd},V_{bs}}}(CLTI_{c,r,V_{dd},V_{bs}}) &= P[CLTI_{c,r,V_{dd},V_{bs}} \leq clti_c] \\ &= \int_{-\infty}^{clti} \int_{-\infty}^{\infty} \frac{1}{|nb_c|} f_{NB_c,T_d}\left(nb_c, \frac{CLTI_{c,r,V_{dd},V_{bs}}}{nb_c}\right) d(nb)\ d(clti) \end{aligned} \tag{6.37}$$

where, $f_{NB_c,T_d}(\cdot)$ is the joint distribution function of two random variables $NB_c(\zeta)$ and $T_d(\zeta)$. As these two random variables are statistically independent, the joint distribution function can be re-written as $f_{NB_c,T_d}(\cdot) = f_{NB_c}(nb_c) \cdot f_{T_d}(t_d)$. Let η be the timing yield constraint of a communication task c. This timing yield constraint gives a limit to scale the voltages (supply and body bias) for the exploitation of slack of each communication task. We further assume that the timing yield is constrained by the range 0.5 $< \eta \leq 1$. The overall delay constraint of a communication task c can be written as,

$$\forall c \in C,\ P(dl_c - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}} \geq 0) \geq \eta \tag{6.38}$$

where, the dl_c is the deadline of each communication task c, $\delta_{i,j}^{\Delta V_{dd}}$ and $\delta_{i,j}^{\Delta V_{bs}}$ correspond to the delay overhead due to supply and body bias voltages, respectively. Eq. (6.38) constraints the probability of a task c having a delay less than the deadline of the task to be more than the confidence level η. From Eq. (6.37), we obtain the distribution function of the data transfer delay CLTI of each task $c \in C$. In practice the distribution function of data size $NB_c(\zeta)$ of each communication task can have any arbitrary distribution function, as a result of this the distribution function of each CLTI can have also any distribution. However, in this work we assume that the delay CLTI has a normal distribution with a mean $\mu_{CLTI}(NB_c, T_d)$ and a standard deviation $\sigma_{CLTI}(NB_c, T_d)$. Thus Eq. (6.38) can be reformulated as follow,

$$\begin{aligned} P(dl_c - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}}) \sim \\ f(\mu_{CLTI}(NB_c, T_d), \sigma_{CLTI}(NB_c, T_d) \geq 0) \geq \eta \end{aligned} \tag{6.39}$$

where, $f(\cdot)$ is the probability density function of $CLTI_{c,r,V_{dd},V_{bs}}$. Eq. (6.39) gives a probabilistic constraint for a communication task c for voltage scaling to reduce the communication energy consumption and to mitigate the effect of process variation. Its

equivalent deterministic constraint can be formulated as,

$$\begin{aligned} &\forall c \in C, \\ &dl_c - \mu_{CLTI}(NB_c, T_d) - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}} - \phi^{-1}(1-\eta) \cdot \sigma_{CLTI}(NB_c, T_d) \geq 0 \end{aligned} \tag{6.40}$$

where, $\phi^{-1}(\cdot)$ is the inverse of the error function. In Eq. (6.40) the selection of supply voltage V_{dd} and body bias voltage V_{bs} for each task depends on the timing yield constraint i.e., the confidence level η. If the yield constraint is set to a high value, supply and body bias voltages are scaled to the minimum level keeping the total delay less than or equal to the deadline dl_c. Eq. (6.40) can be considered as a convex function under the condition that $\eta > 0.5$ as proved in Lemma 6.2.1. Since the target yield for a given communication task is always much higher than 50%, this condition is easily satisfied.

6.3.4 Optimization Algorithm

Similar to **Chap.** 4 and 5 the optimization algorithm presented in this subsection finds the optimal bus width and the number of buses under data size and process variations. The algorithm is based on the cutting hyperplane as presented in **Sec.** 6.2.2.1. In contrast to Problem 6.2.2.1, a simultaneous on-chip communication bus synthesis, supply and body bias voltages scaling under data size and process variations problem is a multi-variable nonlinear optimization problem, which consists of two nonlinear constraints. As in Problem 6.2.2.1, the resulting bus synthesis problem is casted to a convex quadratic optimization problem and solved efficiently using convex optimization tool. We use the interior point method to find the global optimal solution.

Problem 6.3.4.1 *(Combined scheduling, supply and body bias voltages scaling, bus selection, and binding of communication task $c \in C$ to minimize communication bus width and the number of buses with reduced communication energy under data size and process variations) Perform simultaneous scheduling, supply and body bias voltages scaling, bus selection, and binding of each communication task $c \in C$ in order to minimize the communication cost $\sum_{c \in C} Cost_r \cdot b_r$, where b_r is an optimization variable of the bus synthesis problem, subject to: $\forall c \in C$, $P(T_d \leq T_{critical}) \geq \alpha$, $P(dl_c - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}} \geq 0) \geq \eta$, $V_{dd_{min}} \leq V_{dd} \leq V_{dd_{max}}$ and $V_{bs_{min}} \leq V_{bs} \leq V_{bs_{max}}$ for all $t \in \{0 \cdots \lambda\}$, where λ is the maximum possible delay constraint of a communication task $c \in C$ such that the gate delay T_d should be less than equal to the given critical time constraint $T_{critical}$ with a confidence level α for all communication tasks. The second overall delay of a task c should be less than or equal to its given deadline dl_c with a confidence level η. Third and fourth constraints give the limit of supply and body bias voltages scaling.*

The formulation of the simultaneous on-chip communication bus synthesis and supply/body bias voltage scaling problem under process variations is similar to the

formulation presented in **Sec.** 5.4.2.1 except two nonlinear probabilistic constraints for the gate delay and communication tasks. The primary goal of this optimization problem is to minimize the communication cost as given in Eq. (5.11) under data and process variations. While the secondary goals are to minimize communication energy and to mitigate the effect of process variations by scaling supply and body bias voltages, respectively. The binding constraint for each communication task $c \in C$ with bus width r, supply voltage V_{dd}, and body bias voltage V_{bs} is given in Eq. (5.25). The decision variable $X_{c,r,V_{dd},V_{bs}} \in \{0,1\}$ in the binding constraint indicates that each communication task c must be scheduled at time $t \in \{0 \cdots \lambda\}$ with a bus with bus width r, supply voltage V_{dd} and body bias voltage V_{bs}. At time t if more than one communication task c is to be scheduled then separate buses are assigned to them as shown in Eq. (5.27), where a variable b_r is an optimization variable. This constraint avoids any bus conflict among the communication tasks. If there is a dependency between two communication tasks (c_i, c_j), where task c_i being a predecessor and task c_j being a successor then task c_j should not start to transfer data before task c_i completes. This dependency constraint is given in Eq. (5.28). Since the gate delay depends on the different parameters of process variation, it can be critical to the performance of a circuit. This variation in gate delay is controlled by body biasing and accept this variation under certain confidence level α, which can be written as,

$$\forall c \in C,\ P\left(\sum_{V_{bs}=V_{bs_{max}}}^{V_{bs_{min}}} (T_d \leq T_{critical}) \cdot X_{c,r,V_{dd},V_{bs}}\right) \geq \alpha \tag{6.41}$$

Eq. (6.41) guarantees that for each communication task c, its gate delay T_d is less than $T_{critical}$ with confidence level α. If this condition is not fulfilled, the body bias voltage will be increased continuously until the above condition meets. Similarly, the timing yield constraint of each communication task under data size variation can be described as,

$$\forall c \in C,$$
$$P\left(\sum_{r\in R}\sum_{\substack{t=\\ ASAP_c}}^{\Psi}\sum_{\substack{V_{dd}=\\ V_{dd_{min}}}}^{V_{dd_{max}}} ((dl_c - t - CLTI_{c,r,V_{dd},V_{bs}} - \delta_{i,j}^{\Delta V_{dd}} - \delta_{i,j}^{\Delta V_{bs}}) \cdot X_{c,t,r,V_{dd},V_{bs}}) \geq 0\right) \geq \eta \tag{6.42}$$

where at the beginning, the supply voltage is scaled from the minimum value $V_{dd_{min}}$ and the algorithm accepts the data transfer delay CLTI of a task c with a confidence level η. If the overall delay is not less than or equal to the deadline dl_c then supply voltage is increased continuously unless the condition fulfills. Note that when supply voltage is increased, the variation of the data transfer delay CLTI decreases quadratically. Thus the supply voltage is scaled to exploit the slack of communication task c,

while the body bias voltage is applied mainly to mitigate the effect of process variations. For continuous scaling of supply and body bias voltages, their constraints are given in Eqs. (5.18) and (5.30).

Above simultaneous on-chip communication bus synthesis and supply/body bias voltage scaling problem with two nonlinear probabilistic constraints is casted as a convex quadratic optimization problem as in Problem 6.2.2.1. This problem can be solved using any convex optimization tool, which gives a global optimal solution. Since the convex quadratic constraint in Eq. (6.42) is a convex function for timing yield constraint $\eta > 50\%$, it guarantees the global optimal solution as proved in Lemma 6.2.1. As continuous bus selection and continuous voltage selection (CBS-CVS) problem can be solved in a polynomial time complexity, selected voltages can not be applied to a real digital system design. However, both discrete bus selection and continuous voltage selection (DBS-CVS) and discrete bus selection and discrete voltage selection (DBS-DVS) problems are known to be NP-hard, thus we use the heuristic proposed in **Sec.** 5.4.3, which performs voltage selection continuously and chooses their corresponding discrete supply and body bias voltages in a *quasi-polynomial* time complexity. The details of the heuristic is given in Algorithm 5.2. Since the goal is to find the best timing yield constraint so as to find the optimal bus width and the number of buses with reduced communication energy, we use the timing yield constraint search algorthm presented in **Sec.** 6.2.2.2. The search algorithm starts from the maximum value of time yield constraint η and at each iteration, the algorithm looks for the best one that minimizes communication bus cost and energy consumption by mitigating the effects of process variations.

6.3.5 Parameter Estimation of Voltage

In real-time distributed embedded systems, a driver attached to each module is capable to scale supply and body bias voltages under variable workload. The voltage scaling is done dynamically after detecting the load during run time, however in this subsection, we estimate analytically the probability density function of supply voltage V_{dd} and body bias voltage V_{bs}. When the above probabilistic delay models of gate and communication tasks c are applied to the optimization algorithm presented in **Sec.** 6.3.4, the optimal bus width $b_r(opt)$ for each communication task will be obtained. After an algebraic manipulation of Eq. (6.27), the gate delay T_d is inversely proportional to (V_{dd} - V_{th}). We make this approximation in order to render the illustration of our approach more accessible, however the drawn conclusions are valid for the general case. After a simple algebraic manipulation of Eqs. (6.36) and (6.27) we get,

$$CLTI_{c,r,V_{dd},V_{bs}} = \kappa \frac{NB_c(\zeta)}{b_r(opt) \cdot (V_{dd} - V_{th})} \tag{6.43}$$

From Eq. (6.43), the statistical parameters of supply and threshold voltages in terms of the optimal bus width $b_r(opt)$ can be formulated as,

$$(V_{dd} - V_{th})_c = \frac{NB_c(\zeta)}{b_r(opt) \cdot CLTI_{c,r_{opt},V_{dd},V_{bs}}} \tag{6.44}$$

In Eq. (6.44) the data transfer delay $CLTI_{c,r_{opt},V_{dd},V_{bs}}$ is a function of the optimal bus width $b_r(opt)$, the supply voltage V_{dd}, and the body bias voltage V_{bs}. Under variable workload, voltages are scaled to exploit the slack, which reduces the communication energy consumption and mitigates the effect of process variations. As a result of voltages scaling the data transfer delay $CLTI_{c,r_{opt},V_{dd},V_{bs}}$ gets very close to the deadline dl_c of communication task c. Furthermore, the percentage of slack exploitation of each task depends on the timing yield constraint η. Thus in Eq. (6.44) the data transfer delay $CLTI_{c,r_{opt},V_{dd},V_{bs}}$ can be replaced by the timing yield constraint η and results,

$$(V_{dd} - V_{th}) = \mathcal{K}_{V_{dd},V_{th}} \cdot NB_c(\zeta) \tag{6.45}$$

$$\mathcal{K}_{V_{dd},V_{th}} = \frac{\kappa}{b_r(opt) \cdot dl_c \cdot \mathcal{P}_{V_{dd},V_{bs}}} \tag{6.46}$$

$$\mathcal{P}_{V_{dd},V_{bs}} = \frac{\mu_{CLTI}(NB,T_d) + \phi^{-1}(1-\eta) \cdot \sigma_{CLTI}(NB,T_d)}{dl_c - [\mu_{CLTI}(NB,T_d) + \phi^{-1}(1-\eta) \cdot \sigma_{CLTI}(NB,T_d)]} \tag{6.47}$$

where $\mathcal{P}_{V_{dd},V_{bs}}$ is the percentage of slack exploitation, $\mu_{CLTI}(NB,T_d)$ and $\sigma_{CLTI}(NB,T_d)$ are the mean and the standard deviation of the data transfer delay CLTI, respectively. The mean CLTI remains constant for all values of the timing yield constraint, while its standard deviation changes. However, for a fixed timing yield constraint η, the percentage of slack exploitation $\mathcal{P}_{V_{dd},V_{bs}}$ is constant. Thus the joint density function of supply and threshold voltage under variable workload and process variation can be derived as,

$$f_{V_{dd},V_{bs}}(nb,t_d) = \left(\frac{1}{\mathcal{K}_{V_{dd},V_{th}}}\right) f_{NB}\left(\frac{nb}{\mathcal{K}_{V_{dd},V_{th}}}\right) \tag{6.48}$$

The marginal density functions $f_{V_{dd}}(v_{dd})$ and $f_{V_{th}}(v_{th})$ of supply voltage and body bias voltage, respectively, can be evaluated as,

$$f_{V_{dd}}(v_{dd}) = \int_{v_{th_1}}^{v_{th_2}} f_{V_{dd},V_{th}}(v_{dd},v_{th})\, dV_{th} \tag{6.49}$$

$$f_{V_{th}}(v_{th}) = \int_{v_{dd_1}}^{v_{dd_2}} f_{V_{dd},V_{th}}(v_{dd},v_{th})\, dV_{dd} \tag{6.50}$$

In the communication task graph $G(C,\Pi)$ shown in Fig. 6.1, the data size to be transferred by communication tasks $c \in C$ is modeled independent random variables and

each communication task can have an arbitrary distribution function. Thus, the distribution of supply and threshold voltage do not come out to be identical for all communication tasks $c \in C$. The overall density function of supply voltage of all communication tasks is derived by summing the individual density functions of supply voltage V_{dd} as shown in Eq. (6.24). Similarly, the overall density function of threshold voltage of all communication tasks is,

$$f_{V_{th}} = \sum_{j=1}^{|C|} f_{V_{th_j}} \tag{6.51}$$

The sum of the density function of an individual threshold voltage can be computed by convolving the individual density functions,

$$f_{V_{th}} = f_{V_{th_j}} * f_{V_{th_{j+1}}} * \cdots * f_{V_{th_C}} \tag{6.52}$$

After an algebraic manipulation of Eqs. (6.28) and (6.52) the density function of body bias voltage V_{bs} can be obtained.

6.4 Summary

We investigated the effect of process variations on the simultaneous on-chip bus synthesis and voltage scaling in presence of random data size to be transferred among the on-chip modules. The data size and process parameters variations are modeled as a random variable with known probability distribution function. For the first synthesis and modeling approach, an additional weight so called timing yield constraint η was included in the bus synthesis and optimization formulation, which synthesize the optimal bus widths and the number of buses under random on-chip data traffic. Later, the bus synthesis model is extended for combined data size and process variations, where supply voltage is scaled to minimized and the dynamic power, while the body biasing is used for mitigating the effects of process variations.

Chapter 7

Methodology Validation

Contents

The goal of this chapter is to validate the methodology described in previous chapters for synthesizing on-chip communication bus. The synthesis is performed for two different cases: 1) real-time constraints without considering power 2) power aware bus synthesis under deterministic data traffic, random data traffic, and process variation. We assume that a system specification has been partitioned and mapped onto the appropriate modules of an SoC as shown in Fig. 7.1, where the mapped hardware and software are considered to be black boxes. A partitioned hardware and software system is profiled to extract the communication behavior between each on-chip modules. As a result of profiling, a set of communication tasks with their dependencies, data sizes and timing informations are obtained. The extracted informations are applied for the on-chip communication bus synthesis.

This chapter is organized as follows. At first **Sec.** 7.1 describes different benchmarks which are used in this chapter to validate the proposed bus synthesis technique. **Sec.** 7.2 introduces a profiling technique to extract communication tasks, data size, and

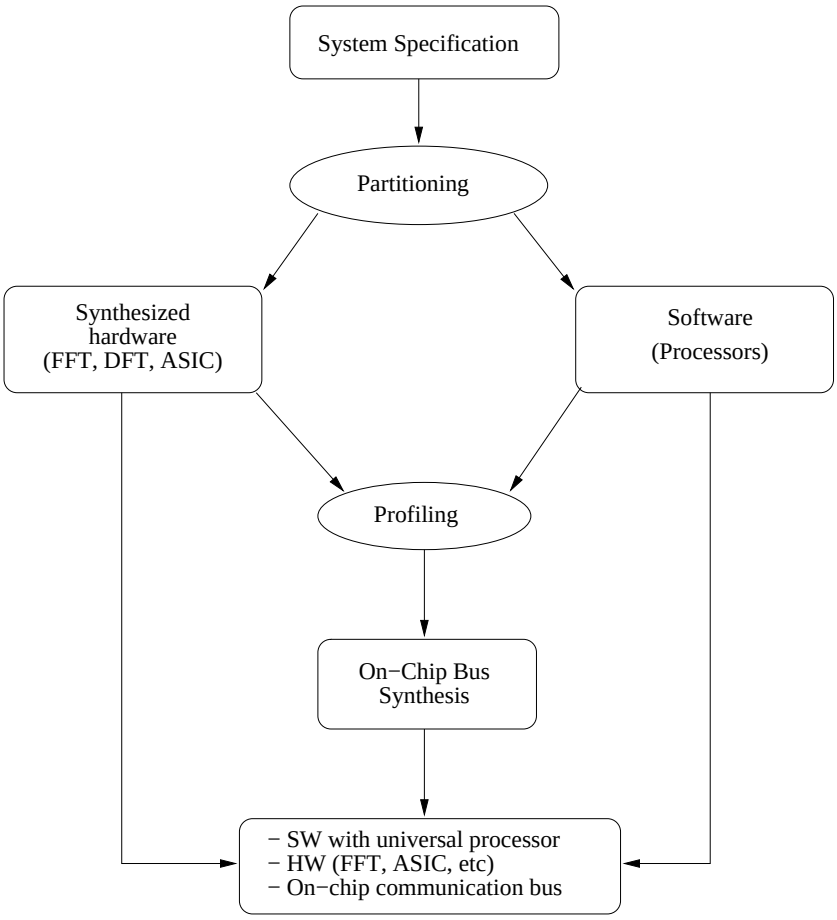


Fig. 7.1: Design flow for a mixed hardware/software system

timing. **Sec.** 7.3 presents results of applying bus synthesis techniques discussed in **Chap.** 4, 5, and 6. Finally, **Sec.** 7.4 gives a summary of this chapter.

7.1 Benchmarks

7.1.1 Real-life Applications

Ogg Vorbis Decoder: Ogg Vorbis is an audio compression format developed by the Xiph.org Foundation [9], which is a non-profit organization working to provide a free multimedia technology. Ogg is a large framework for several multimedia applications including Vorbis (audio) and Tarkin (video). It defines a data format to be packed into streams and transported regardless of data content in the stream which can be Vorbis or Tarkin. Ogg bit-streams are streams of octets which can compose of several logical streams inside one physical stream using a multiplexing or chaining techniques.

Ogg Vorbis is a lossy, asymmetrical algorithm and uses several techniques such as

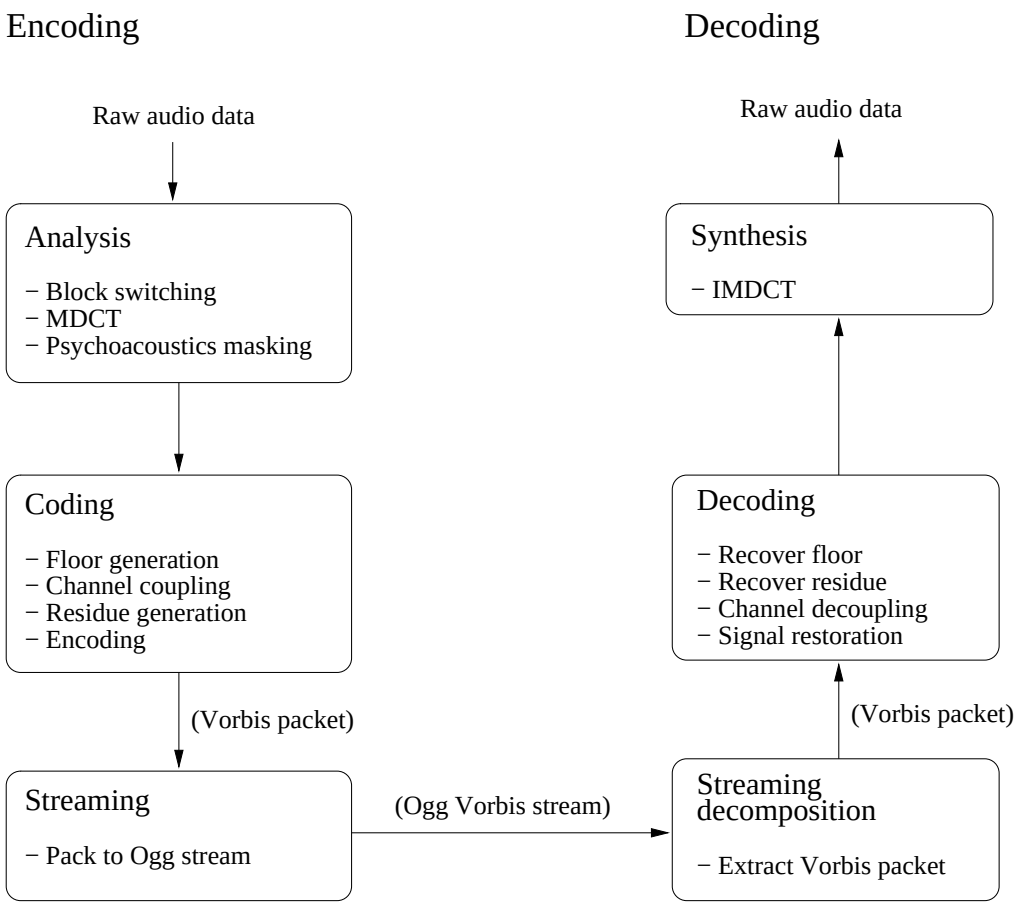


Fig. 7.2: Ogg Vorbis encoding and decoding technique [9]

dividing of input into short blocks, MDCT (modified version of DCT), psychoacoustics, vector quantization, predictive, and many other for the audio encoding and decoding. In Ogg platform, an audio data is encoded by the Vorbis encoder, packed into an Ogg bit stream and then send to Vorbis decoder. Vorbis decoder opens the Ogg bit stream, unpack the Vorbis stream data, decode and provide an uncompressed raw audio data.

Ogg Vorbis encoding and decoding stages are shown in Fig. 7.2. In general, Vorbis encoding and decoding are done in six different stages [9]. Vorbis decoder takes a raw audio data as overlapped but contiguous short-time segments and analyzes the audio data to find the optimal small representation, which is called analysis. In this stage, audio data is divided into overlapping blocks of two sizes: short (256 samples) or long (2048 samples). For the exception case with unusual sound or noise a long window size is used to avoid their effects. The step is called block switching, where each block is transformed into frequency domain using MDCT and then analyzed the psuchoacoustics masking step. Then the audio data is encoded into a much smaller data representation as determined in the previous step. This stage is called coding,

where information received from psychoacoustics masking process is used to create the spectral envelope of the signal and floor functoin [9]. Small representation of audio data (floor and residue) are encoded using VQ (vecotr quantization) to form a vorbis packet. Once the data is coded, raw audio data packets are packed into streams, called streaming. While for decoding the audio streams at the decoder, it first extracts the sequence of raw packets from the stream and this technique is called streaming decomposition. Then the decoder reconstruct the sound signal representation from these received audio packets, called decoding stage. After the decoding stage, audio data of frequency domain is transformed into time domain using inverse MDCT, which is the last stage of the Vorbis decoding.

Sphinx Speech Recognition: In recent year speech recognition technique has emerged a solution to the problems of human-computer interaction. It is a process of converting an acoustic signal, captured by a microphone or a telephone to a set of words. The recognized word(s) can be used later for several applications such as commands, controls, and data entry. The recongition is mainly based on the hidden Markov model (HMM), which represents a possible symbol sequences underlaying speech utterances. Fig. 7.3 shows the data flow of the Sphinx [8] speech recognition system. Training takes as input a large number of speech utterances along with their transcriptions into phonemes and outputs the speech models for the phonemes. The utterences to be recognized first undergo a spectral analysis stage, also called the feature extraction stage. Typical feature representations are smoothed spectra or linear prediction coefficients.

As a main task of speech recognition system, it takes a given observation sequences $V = O_1, O_2 \cdots O_n$ (each O_i represents a feature vector), and a set of HMMs, (each HMM represents a phoneme), the decoder tries to find the model (M) that best matches the observation sequences, $P(O|M)$, given the model M is maximized. An N-stage Markov model is defined by a set of N states forming a finite state machine such that $a_{i,j}$ is the transition probability from state i to j. Each state is additionally associated with a probability density function $b_j(O_t)$ representing the probability that a particular observation O_t is emitted by state probability j for observation number t. These probability are estimated during training.

The probability $P(O|M)$ is approximated by the probability of the state sequence Q maximizing $P(O, Q|M)$. For a given model M, let $\psi_j(t)$ represents the maximum likelihood of having observed the sequence O, and being in state j at time t. This partial likelihood can be computed as [8]:

$$\psi_j(t) = max_i\{\psi_i(t-1)a_{i,j}\}b_j(O_t) \tag{7.1}$$

The maximum likelyhood $P_m(O|M)$ is then given by $\psi_N(n) = max_i\{\psi_i(n)a_{iN}\}$. The spoken utterances modeled by HMMs are sub-word constructions called phonemes, while words are chains of phonemes. The word models are then aggregated using a language model as shown in Fig. 7.3.

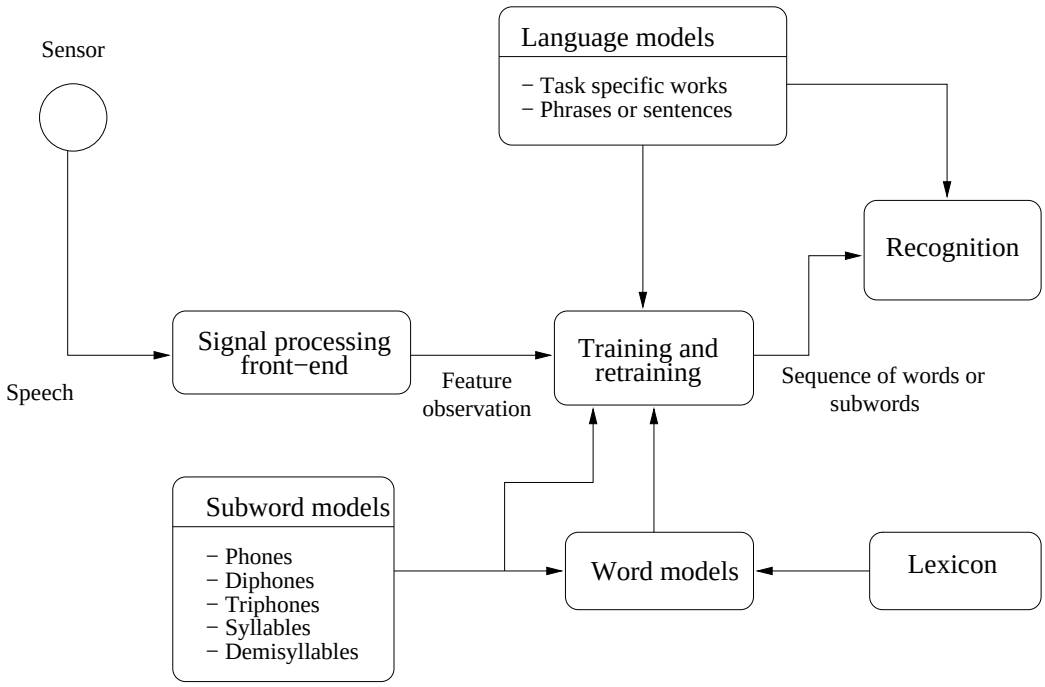


Fig. 7.3: Sphinx speech recognition system [8]

7.1.2 Randomly Generated Tasks

A randomly generated tasks consists of 119 communication tasks. The parameters such as task dependency, data size, and timing are randomly assigned to each communication tasks.

7.2 Profiling

In order to trace the communication among the on-chip communicating modules, a software profiling tool is used that finds communication tasks, dependency, data size, and the timing. This technique is based on the static profiling so that the on-chip communication behavior is extracted for different scenarios before synthesizing a bus. In this work, the GNU profiler called *gprof* is used to get the detail information about the functions call and their run time. While a Sparc processor based platform (*sparc-sun-solaris*) is used for cross compiling a hardware/software system. There are three main steps for profiling a system, which are as follows:

- cross compile a system specification with profiling enabled for a target platform (in this work *sparc-sun-solaris* is used)
- execute it to generate a profile data file

- run *gprof* to analyze the profile

With profiling we can analyze when a function is called and which functions was called by a set of other functions as shown in Fig. 7.4. The main commands used for generating such a tree is given as follows:
$ env CFLAGS=-pg LDFLAGS="-pg -static" ./configure -target=sparc-sun-solaris -prefix=/home/apps/SystemOnTest
$ make
$ SystemOnTest/bin/<sample> SystemOnTest/<dir> SystemOnTest
$ gprof SystemOnTest/bin/<sample> <sample>.gmon > SystemProfile.txt

In the first line the *-pg* option compiles and links a system specification with profiling support, the *-target=sparc-sun-solaris* option sets the object platform of the cross compilation. By this configuration the C compiler in the makefile is set as *sparc-sun-solaris-gcc* and the source code of a system is cross compiled with a C compiler *sparc-sun-solaris-gcc*. In the second line the source code is cross compiled to the target directory *apps/SystemOnTest*. In the third line a source code is executed for a given input. The fourth line profiles an executable file and collect the profiled information in a text file, which consists of set of functions, its run time, and calling relation. After profiling a system a full tree structure of called functions is shown in Fig. 7.4. In the figure a function at the tail of an arrow calls a function at the head of the arrow.

Table 7.1 gives the timing information of called function. The term parents is the name of caller function and the children is the name of functions which are being called by caller (parents). The meaning of the fields in the primary line is given as follows:

- **name:** this is the name of current function
- % **time:** is the percentage of total time that was spent in this function, including time spent in subroutines called from this function.
- **self:** means the total amount of time spent in a function.
- **descendents:** the total amount of time spent in the subroutine calls made by this function. This should be equal to the sum of all the self and children entries of the children listed directly below this function.

7.3 Bus Synthesis

To validate the proposed methodology for on-chip bus synthesis, we use two benchmarks randomly generated tasks and real-life applications throughout this chapter.

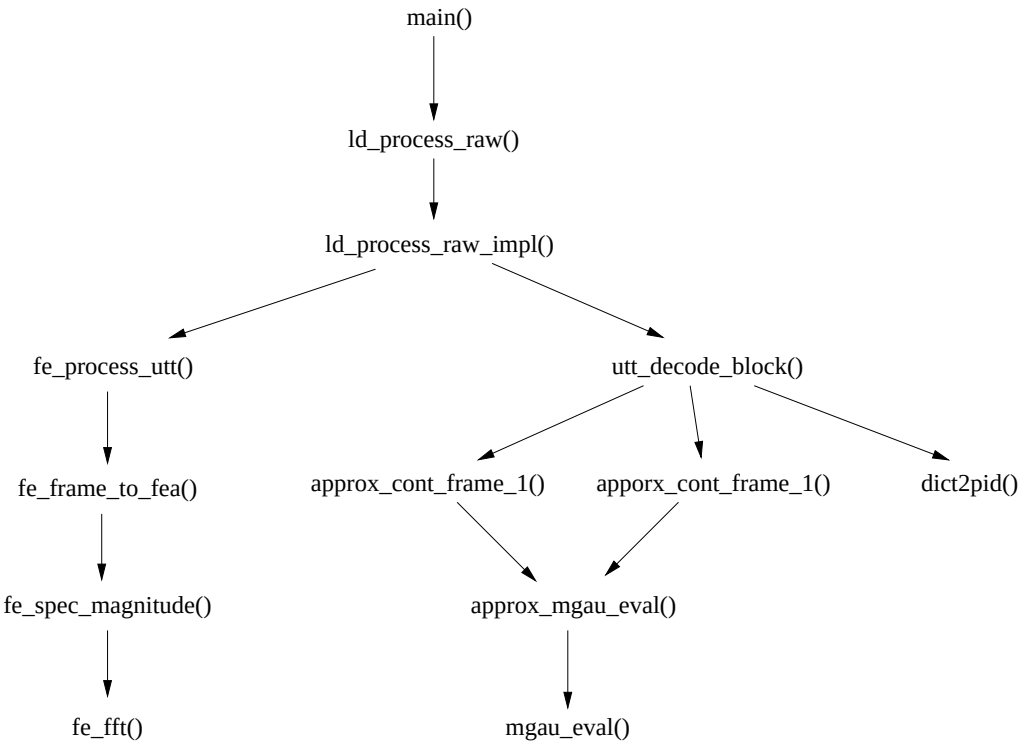


Fig. 7.4: Full tree structure of functions call

			parents
% time	self (ms)	descendents (ms)	name
			children
	0.00	0.11	ld_end_utt
	0.00	11.54	ld_process_raw
81.6	0.00	11.64	ld_process_raw_impl
	0.03	11.10	utt_decode_block
	0.03	0.47	fe_process_utt
	0.00	0.00	_ckd_calloc_2d_

Tab. 7.1: Information of a called graph with their timing

The real-life application includes Ogg Vorbis decoder and Sphinx speech recognition system. As discussed in **Sec.** 7.1, Ogg Vorbis decoder includes four main decoding steps which are inverse quantization, channel decoupling, reconstruct curve, and IMDCT. After manually partitioning and mapping of Ogg Vorbis decoder, the IMDCT is mapped to a single hardware and rest of the functionalities are mapped to a processor. Furthermore, incoming Ogg Vorbis data is mapped to a compact frash (CF) memory with an CF-interface and the extracted audio data are mapped to an audio buffer. Similarly, the second one CMU Sphinx [8] open source for speech recognition application, which consists of three main components: front end, decoder and linguist. The front end includes series of data processing tasks such as pre-emphasis, hamming window, FFT (fast fourier transformation), mel frequency filter, IFFT, cepstral mean normalization, and feature extraction to generate the features from the speech. The training takes as input a large number of speech along with their transcriptions into phonemes to provide the speech models for the phonemes. The recognition is based on the HMM (hidden markov model) to decode the speech. We used the American English lexicon consisting of 32 phonemes and a database of 17 different words (spelling out the names of the months, numbers and digits). The length and the number of phonemes in a speech varies from application to application. After partitioning, the front end was mapped to dedicated hardware including FFT and filters. The task training and recognition were mapped to a PowerPC processor. We profiled the C model of the Ogg Vorbis with 38 seconds of audio data [194] and extracted 94 communication tasks with their timing timing and data size. The real time constraint for a session is evaluated as $T_{session}$ = 9.96 μs for the audio application. Similarly, the C-model of the Sphinx speech recognition algorithm was profiled [193] and extracted 119 communication tasks with timing and data size.

The on-chip communication buses were given as a library of buses with different bus widths, which ranges from 16 to 128-bit with an increment of 4-bit. For the experiment purpose, we consider a bus with $4mm$ in length and its corresponding single line capacitance for $0.07\mu m$ technology is $609fF$ as estimated in [107]. Furthermore, the load capacitances of driver C_{dri} and receiver C_{rec} per single line are $13fF$ and $10fF$, respectively. The bus synthesis algorithm was implemented in C as a pre-processing model to interface with a convex solver of the MOSEK [5] and run on a 1.5 GHz Intel P4 PC with 256 kb cache. The bus synthesis algorithm is shown in Algorithm 7.1, where from line 1-21, it reads inputs data such as library of buses, voltages, and technological dependent parameters. From line 23-26, the algorithm computes the minimum delay to transfer data for each tasks. At line 27, 28, and 29, scheduling, allocation, and binding subroutines are called. These subroutines are depicted in Algorithm 7.2, 7.3, and 7.4 respectively.

```
BUSSYNTHESIS()
 1  Tasks ← GETTASKS();
 2  PossibleStartTime ← GETPBLTIME();
 3  MaxBusSize ← GETMAXBUSSIZE();
 4  MaxVolt ← GETMAXVOLT();
 5  Vth0 ← GETVTH0();
 6  NumBusTypes ← GETNUMBUSTYPES();
 7  NumVoltLevel ← GETNUMVOLTLEVEL();
 8  Eta ← GETETA();
 9  DeltaVdd ← GETDELATAVDD();
10  Sigma3Vth ← GET3SIGMAVTH();
11  K1 ← GETTECHPARAK1();
12  DIBL ← GETTECHPARADIBL();
13  K2 ← GETTECHPARAK2();
14  Depn ← GETDEPN();
15  BusLib ← GETBUSLIB();
16  Vdd ← GETVDD();
17  Vbs ← GETVBS();
18  DataSize ← GETDATASIZE();
19  SigmaData ← GETSIGMADATA();
20  ASAP ← GETASAPTIME();
21  ALAP ← GETALAPTIME();
22  /*Computes the minimum delay to transfer data*/
23  for (i = 0; i ≤ Tasks; i++)
24  do
25      grossData ← DataSize[i] + sigma[i];
26      MinDelay[i] ← CALDELAY(grossData, MaxBusSize, MaxVolt);
27  SCHEDULING();
28  ALLOCATION();
29  BINDING();
30  return ;
```

Algorithm 7.1: Bus synthesis algorithm.

```
SCHEDULING()
 1  for (i = 0; i ≤ Tasks; i++)
 2  do
 3      for (succ = 0; succ ≤ Tasks; succ++)
 4      do
 5          (depn[i][succ] == 1)
 6          for (j = 0; j ≤ NumBusTypes; j++)
 7          do
 8              for (k = 0; k ≤ NumVoltLevel; k++)
 9              do
10                  for (z = 0; z ≤ NumVbsLevel; k++)
11                  do
12                      (gdPre, gdSucc) ← COMPUTEGROSSDATASIZE(pre, succ);
13                      (stDelayPre, stDelaySucc) ← CALSTATDELAY(pre, succ);
14                      for (t = 0; t ≤ ALAP[succ] − ASAP[succ] + delaySucc; t++)
15                      do
16                          COMPUTEVARIABLES();
17
18                      for (t = 0; t ≤ ALAP[i] − ASAP[i] + delayPre; t++)
19                      do
20                          COMPUTEVARIABLES();
21                          return ;
```

Algorithm 7.2: Algorithm for scheduling of communication tasks.

```
ALLOCATION()
 1  for (i = 0; i ≤ Tasks; i++)
 2  do
 3      for (j = 0; j ≤ PossibleStartTime; j++)
 4      do
 5          for (k = 0; k ≤ Tasks; k++)
 6          do
 7              for (v = 0; v ≤ NumVoltLevel; v++)
 8              do
 9                  for (z = 0; z ≤ NumVbsLevel; z++)
10                  do
11                      grossData ← COMPUTEGROSSDATASIZE(i);
12                      delay ← CALSTATDELAY(i);
13                      for (b = j − delay; b ≤ j; b++)
14                      do
15                          for (a = 0; a ≤ ALAP[k] − ASAP[k] + delay; a++)
16                          do
17                              if (b == a)
18                                then
19                                      COMPUTEVARIABLE();
20                                      return ;
```

Algorithm 7.3: Algorithm for allocation of communication tasks.

```
BINDING()
 1  /*Evaluates the binding constraints*/
 2  for (i = 0; i ≤ Tasks; i++)
 3  do
 4      for (j = 0; j ≤ NumBusTypes; j++)
 5      do
 6          for (k = 0; k ≤ NumVoltLevel; k++)
 7          do
 8              for (z = 0; z ≤ NumVbsLevel; k++)
 9              do
10                  delay ← CALSTATDELAY(grossData, BusLib, Vdd, eta, Vbs)
11                  for (t = 0; t ≤ ALAP − ASAP + delay; t++)
12                  do
13                      COMPUTEVARIABLE();
14                      return ;
```

Algorithm 7.4: Algorithm for binding of communication tasks.

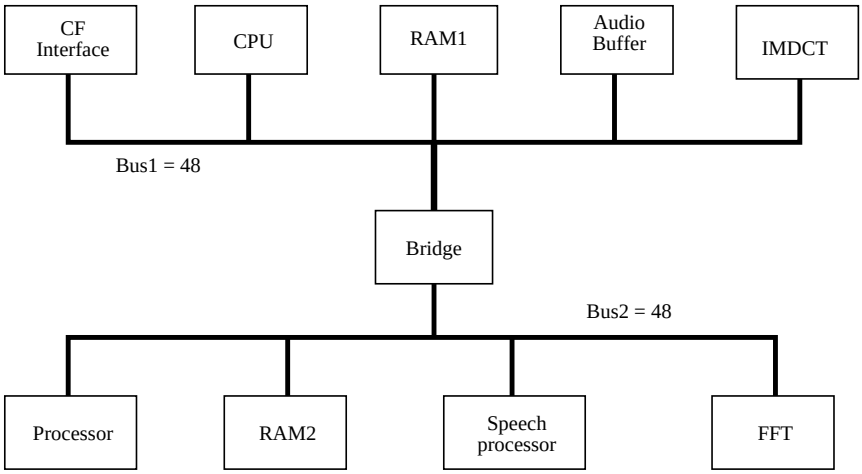


Fig. 7.5: The synthesized bus architecture for Ogg Vorbis and speech recognition systems

7.3.1 Real-time Constraints

Based on the mixed NLP formulation described in Sec. 4.2.2, we conducted an experiment to find the optimal bus width and the number of OCTs using a commercial optimization tool MOSEK [5]. The total of 213 communication tasks were provided to the optimizer with the hardware constraint of the bus width $16 \leq b_r \leq 64$ bit wide buses. We found the optimal bus width of 48 bit wide with number of four OCTS for the real-time constraint of 9.93 μs. The MOSEK tool took about 3.46 minutes to obtain this result on a 1.5 GHz Intel P4 PC with 256 kb cache on the Microsoft Windows platform.

In Tab. 7.2, it can be seen that changing bus width, the number of OCTs $\mathcal{N}_o$ and $\mathcal{N}_\kappa$ of the CLTIs also change. These changes have a nonlinear behavior in respect to the bus width b_r. This nonlinearity is due to the time constraint w (minimum processing time of a processor) between predecessors and successors as shown in Fig. 4.2. In addition to this, in the fifth column the total duration of OCTs $(\mathcal{D}_o + \mathcal{D}_\kappa)$, which is in this case decreasing with an increasing bus width. After the scheduling of tasks, we applied the clique partitioning algorithm to find a communication topology. As a result, we obtain two 48-bit wide communication buses with their corresponding interconnections with the communication. We further optimized the synthesized communication topology by evaluating the total communication cost of each communication task. For this purpose, the maximum number of bridge accesses $MaxNumOf_{req}$ and, the maximum burst size, the cost C_δ with, and the cost C_s are set to 34, 8, 4.25 and 1, respectively. These parameters were considered for the worst case and they were adapted from the ABMA [1] on-chip communication bus protocol. Tab. 7.3 depicts the total communication cost of each on-chip module and their possible swap which indicates if it is possible to swap the module to another bus. Where the FFT has the maximum cost and the

BusWidth(r)	$(\sum CLTI + w)_{sesion}$	$\mathcal{N}_o$	$\mathcal{N}_c$	$\sum(\mathcal{D}_o + \mathcal{D}_\kappa)\mu$s
16	15.04 μs	19	17	76.75
20	13.98 μs	21	16	62.63
24	12.72 μs	13	14	52.53
28	12.13 μs	13	12	44.40
32	11.74 μs	12	8	37.91
36	11.04 μs	10	6	32.72
40	10.64 μs	6	6	28.84
44	10.02 μs	4	3	25.85
48	**9.93 μs**	**4**	**3**	**23.32**
52	9.84 μs	3	5	21.16
56	9.77 μs	3	5	19.28
60	9.73 μs	3	4	17.64
64	9.72 μs	3	3	16.45

Tab. 7.2: Number of OCTs among the communication tasks for different bus widths

Module	A_{comm}	δ	S	Cost	Bus1	Bus2	Swap
IMDCT	0.63	0.21	0.67	2.19	$\surd$	–	–
Audio buffer	0.24	0.24	0.13	1.39	$\surd$	–	$\surd$
CPU	0.48	0.47	0.43	2.9	$\surd$	–	–
Speech processor	0.83	0.86	0.27	4.45	–	$\surd$	–
FFT	0.87	0.72	0.53	4.75	–	$\surd$	$\surd$
CF-interface	0.57	0.24	0.72	2.31	$\surd$	–	–
Location processing	0.84	0.66	0.37	4.01	–	$\surd$	–
RAM1	–	–	–	–	$\surd$	–	–
RAM2	–	–	–	–	–	$\surd$	–

Tab. 7.3: The intermodule communication profile of communication tasks and their communication cost

audio buffer has the minimum cost. The synthesized and optimized communication architecture is shown in Fig. 7.5, where the CF-interface, CPU, IMDCT, audio buffer to I/O, and shared memory (RAM1) are assigned to *Bus1*, while the speech processor, location processor, FFT, and the memory (RAM2) are assigned to *Bus2*. There is a bridge in between *Bus1* and *Bus2* in order to establish communication among modules of *Bus1* and *Bus2*.

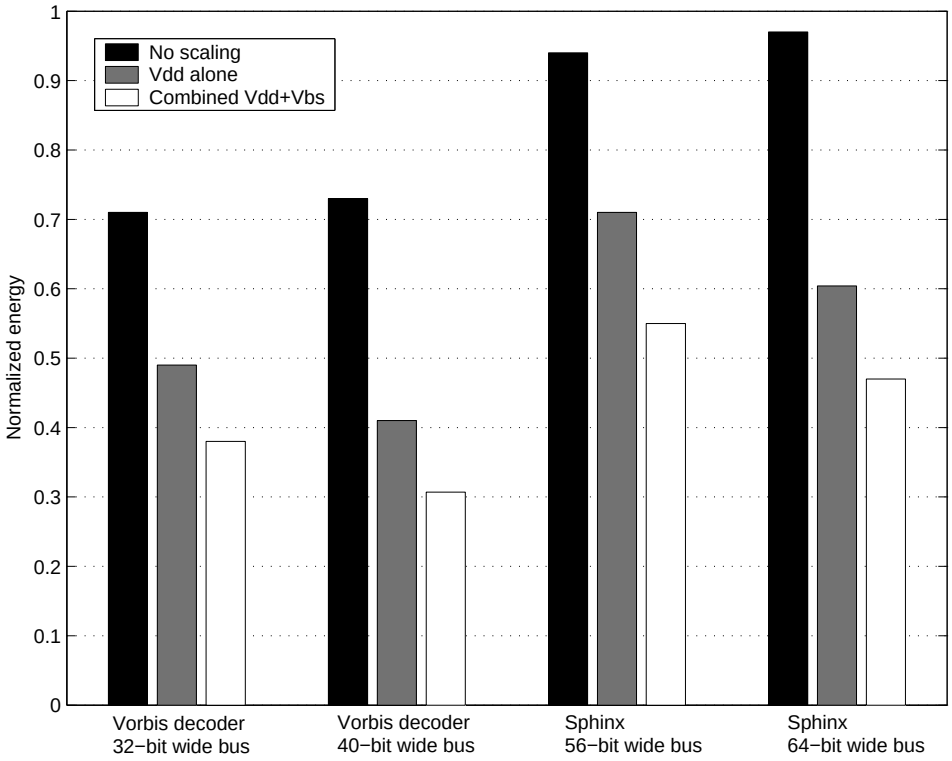


Fig. 7.6: Normalized energy consumption for different synthesized bus width using examples

7.3.2 Simultaneous Bus Synthesis and Voltage Scaling

7.3.2.1 Deterministic Data Traffic

This section presents experimental results to evaluate the effectiveness of the proposed algorithm, which synthesizes an energy efficient on-chip communication bus using supply and body bias voltage scaling techniques. The experiments were conducted on two example systems namely Ogg Vorbis decoder and Sphinx speech recognition system. At first we conducted simultaneous on-chip communication bus synthesis and continuous voltage scaling with an aim to synthesize an optimal bus width and number of buses with reduced communication energy. Voltages of each communication task c were scaled continuously to get the minimum possible energy consumption using the NLP formulation presented in **Sec.** 5.3 and 5.4.2.1.

The supply and body bias voltages were scaled continuously with ranges [1.8V, 0.6V] and [0V, -1V], respectively. The optimal bus width for Ogg Vorbis decoder was 32-bit, while for the Sphinx speech recognition 56-bit. The optimization time for Ogg Vorbis and Sphinx are shown in Tab. 7.4 using the interior point method on a 1.5 GHz Intel P4 PC with 256 kb cache on the Microsoft Window platform.

Fig. 7.6 depicts the total energy consumption of communication buses for different

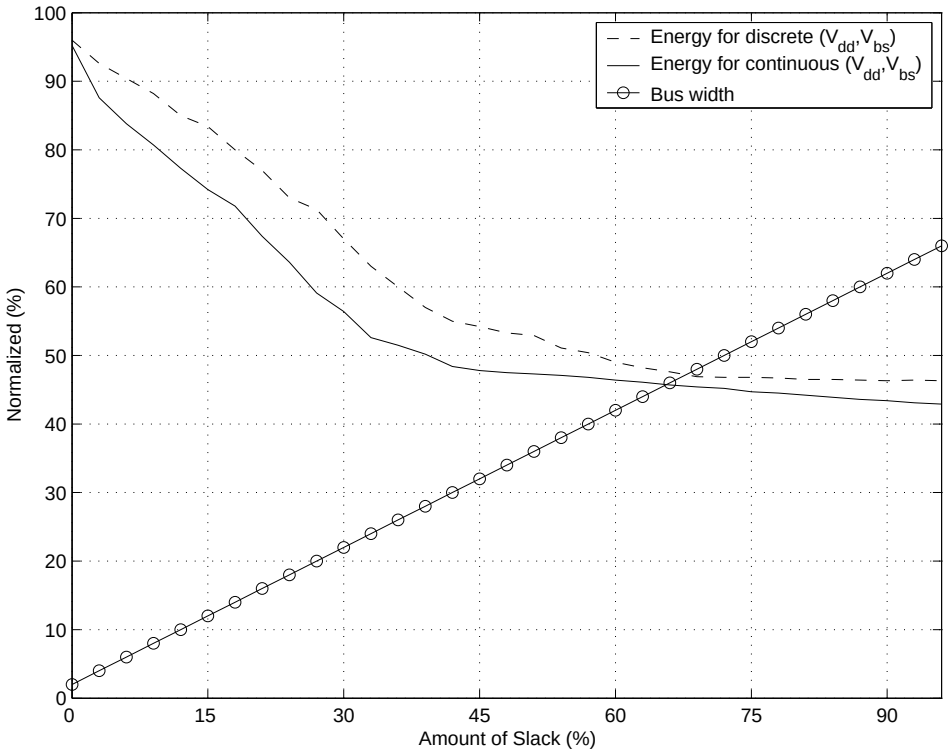


Fig. 7.7: Continuous and discrete voltage scaling for Ogg Vorbis decoder

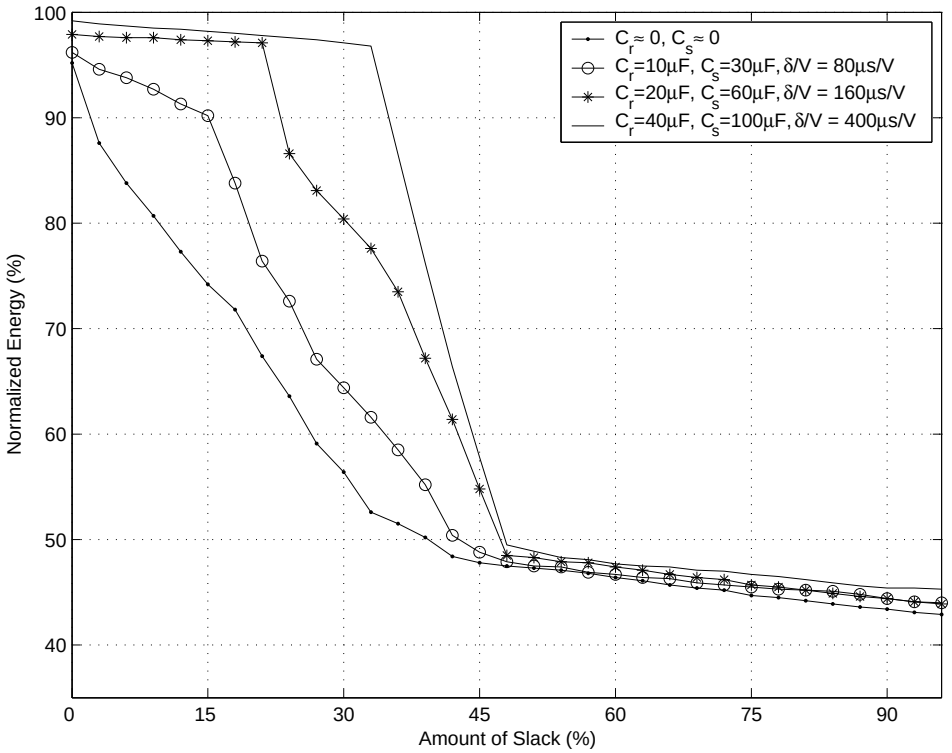


Fig. 7.8: Effect of overhead on energy consumption using voltage scaling

Benchmark	BusWidth	Slack	Run time in (sec.)
Vorbis	32 bit	0 %	17
	40 bit	24 %	31
Sphinx	56 bit	0 %	28
	64 bit	17 %	39

Tab. 7.4: Total amount of slack increment for different bus widths

examples and schemes, which are for nominal voltage scaling, supply voltage scaling, and both supply and body bias voltage scaling. It can be observed that the Ogg Vorbis decoder at its optimal bus width (32-bit wide), the total energy consumption can be reduced by 46.8% when scaling both supply and body bias voltages. This energy can be further reduced by 57.1%, if the next near-optimal bus width is selected, which is 40-bit wide. When bus width is increased from 32 to 40 bit, the amount of slack is increased by 24%, which results in further decrement in commmunication energy consumption. Similarly, for the Sphinx speech recognition unit, the total energy consumption of the bus is reduced by 44.1% at its optimal bus width 56-bit. We further investigated the communication energy consumption at the next near-optimal bus width, which is 64-bit wide. At this near-optimal bus width, the amount of available slack is 17% more than the optimal and this results in decrement in the total communication energy consumption by 52% as shown in Fig. 7.6. When bus width is increased from the optimal solution to increase the slack, the total line capacitace due to wire increases linearly but the quadratic decrease in voltage still causes a significant reduction in energy consumption. However, the cost we have to pay for increasing bus width is an increased in chip size. This justifies that a small increase in chip size may result in reduction of the total energy consumption.

Although continuous voltage scaling techniques gives better run time complexity and energy consumption than discrete voltage scaling, it cannot be used for a digital system design due to its practical limitations[1]. In the second part of experiment, we performed simultaneous communication bus synthesis and discrete voltage scaling to find an optimal bus width and number of buses with reduced communication energy. For discrete voltage scaling, the corresponding supply and body bias voltages are V_{dd_z} = $\{1.8V, 1.4V, 1.0V, 0.6V\}$ and V_{bs_z} = $\{0V, -0.2V, -0.6V, -1.0V\}$. Fig. 7.7 depicts the results of communication bus synthesis and discrete voltage scaling for the Ogg Vorbis decoder with 90 communication tasks. We increased the amount of slack of each communication task c by increasing the bus width. It can be seen that the normalized energy consumption for both continuous and discrete voltage scaling deceases with an increasing amount of slack. However, they remain almost constant for the slack greater than 60%. This is due to the fact that supply and body bias voltages are constrained by

[1] it is very difficult to build a voltage regulator with a small precision

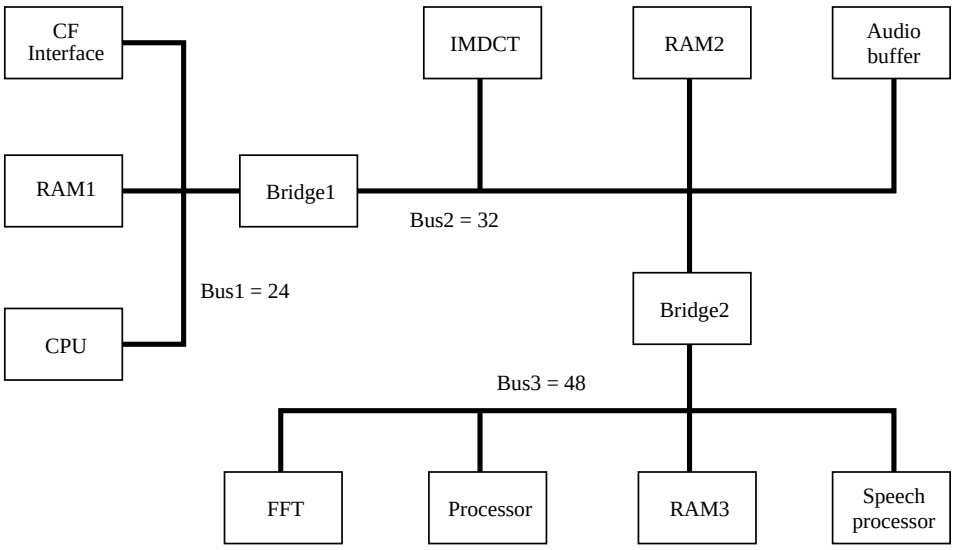


Fig. 7.9: Synthesized an energy efficient bus architecture for an application with Ogg Vorbis and speech recognition

their limits and they cannot be scaled beyound those limits. Furthermore, normalized energy consumption of the continuous voltage scaling technique is less than the discrete voltage scaling technique. Their difference in energy is relatively large for slack less than 60%. In this range, the amount of slack is small and the probability of selecting the next smaller value of supply and body bias voltages is less due to the given deadline of communication task. While for slack greater than 60% the difference is less due to large amount of slack and there is a high probability to select next smaller value of supply and body bias voltages.

In order to investigate an effect of voltage switching overhead on energy consumption, we conducted an experiment on the Ogg Vorbis decoder with 90 communication tasks using discrete supply and body bias voltage scaling technique. The effect of switching overhead was characterized by three parameters, which are power rail capacitance C_r, the total substrate capacitance C_s, and the rate of change of delay with respect to voltage δ/V. Fig. 7.8 depicts the normalized energy consumption versus the amount of slack for different values of switching overhead parameters. As expected, the energy consumption increases for higher values of the parameters determining overhead. For an ideal system with no effect of switching overhead delay caused by overhead switching (capacitances C_r and C_s are zero), the minimum energy consumption can be achieved, that means an energy reduction by 56% if 30% of slack is avail-

able. However, for non-ideal systems with C_r = 20μF, C_s = 60μF, and δ/V = 160$\mu s/V$ achives energy consumption of about 20%.

7.3.2.2 Random Data Traffic

We evaluate the effectiveness of the proposed techniques using a randomly generated benchmark as well as a real-life application, namely speech recognition system. The automatically generated benchmark consists of 119 communication tasks c and the amounts of data to be transferred by all tasks c are normally distributed with the mean $\mu_c(NB)$ = 64 and 128-bit. Different levels of variability in data size $NB_c(\zeta)$ were explored ranging from 2% to 30% of $3\sigma_{NB}$. The deadline dl_c of each task c is deterministic and it is different for different values of σ_{NB}. The data processing time w of each task τ is given for each pair of communication tasks. We evaluate it using Eq. (4.1) for each task τ assuming that on-chip modules are capable to scale the voltages for the variable load [71, 107, 169]. Each communication task c can scale the supply voltage ranging from 1.8V to 0.6V to meet the desired timing yield constraint.

We performed the simultaneous voltage scaling, bus selection, scheduling, and binding of communication tasks c using the proposed algorithm. Tab. 7.7 shows the results of optimized bus width and the number of buses for the automatically generated tasks c. The table compares the bus widths and number of buses $b_r(opt)$, the mean voltage (analytical mean voltage $\hat{\mu}_{V_{dd}}$ and mean of voltage from the Monte Carlo simulation $\mu_{V_{dd}}$), and the analytical mean slack $\hat{\mu}_{Slack}$ for different timing yield constraints η. The results show that the optimized bus width and number of buses vary with timing yield constraint η. In column 2 and 6 of the table, the bus width $b_r(opt)$ is constant for two different values of η, however, the mean voltages $\hat{\mu}_{V_{dd}}$, and $\mu_{V_{dd}}$ and the mean slack $\hat{\mu}_{Slack}$ decrease in column 7, 8, and 9, respectively. This is due to the increase of the timing yield constraint of communication tasks from 79% to 89%. In column 10 of the table, there are two seperate buses with different bus widths for all values of $3\sigma_{NB}$. In this case, the timing yield constraint η of all tasks is set to 99%, such that the voltage of all tasks c is scaled to the minimum possible value. This results in a very small amount of slack of communication tasks. Note that the higher the amount of slack, more the mobility of communication tasks c, which in turn increases communication bus sharing. Hence, at the timing yield of 99%, there is very low mobility of communication tasks c and results in an overlap among them so that two seprate buses are needed to meet the real-time constraints. Fig. 7.15 depicts, the analytical estimation of probability density function (PDF) and cumulative distribution function (CDF) of voltage for η = 79%, $3\sigma_{NB}$ = 20% and $b_r(opt)$ = 48-bit. In Fig. 7.15(a) the density function of supply voltage has a normal distribution with mean voltage 1.1V and its corresponding CDF is shown in Fig. 7.15(b) for voltage range V = 0 to 2.5V. Similarly, Fig. 7.11(a) and (b) depict, the analytical estimation of a probability density function and a cumulative dis-

$3\sigma(NB)$	Timing yield η=79%				Timing yield η=89%				Timing yield η=99%				Run time (sec.)
	$b_r(opt)$ (bit)	$Ana.$ ($\hat{\mu}_V$)	MC (μ_V)	$\hat{\mu}_{Slack}$ (%)	$b_r(opt)$ (bit)	$Ana.$ ($\hat{\mu}_V$)	MC (μ_V)	$\hat{\mu}_{Slack}$ (%)	$b_r(opt)$ (bit)	$Ana.$ ($\hat{\mu}_V$)	MC (μ_V)	$\hat{\mu}_{Slack}$ (%)	
3σ=30%	64	0.93	0.89	59.4	64	0.81	0.80	41.1	(48,24)	0.76	0.71	11.2	$\sim$ 78
3σ=27%	64	0.97	0.94	58.2	64	0.81	0.79	38.7	(48,24)	0.76	0.74	9.5	$\sim$ 81
3σ=25%	60	1.02	0.99	57.5	60	0.81	0.80	38.3	(36,32)	0.76	0.72	9.7	$\sim$ 83
3σ=22%	60	1.07	1.02	57.3	60	0.89	0.85	35.9	(36,32)	0.79	0.75	9.3	$\sim$ 86
3σ=20%	56	1.10	1.04	55.8	56	0.89	0.86	35.6	(32,32)	0.79	0.69	9.8	$\sim$ 89
3σ=17%	56	1.15	1.09	54.4	56	0.92	0.86	34.5	(32,32)	0.84	0.79	7.9	$\sim$ 93
3σ=15%	48	1.19	1.16	54.8	48	0.92	0.87	34.8	(32,32)	0.88	0.81	7.0	$\sim$ 95
3σ=12%	48	1.24	1.15	51.1	48	1.12	1.06	32.6	(32,32)	0.92	0.87	7.1	$\sim$ 97
3σ=10%	36	1.27	1.25	49.3	36	1.12	1.07	32.9	(32,16)	0.97	0.93	6.3	$\sim$ 97
3σ=7%	36	1.32	1.26	49.7	36	1.23	1.14	31.3	(32,16)	1.03	0.94	6.4	$\sim$ 101
3σ=5%	32	1.33	1.28	48.6	32	1.12	1.07	31.4	(32,16)	1.10	1.03	6.5	$\sim$ 113
3σ=2%	32	1.35	1.25	48.1	32	1.27	1.21	30.2	(32,16)	1.13	1.06	6.6	$\sim$ 117

Tab. 7.5: Synthesize bus(es) and bounds on mean voltage for different timing yield constraint (η) and standard deviation (σ) of data size

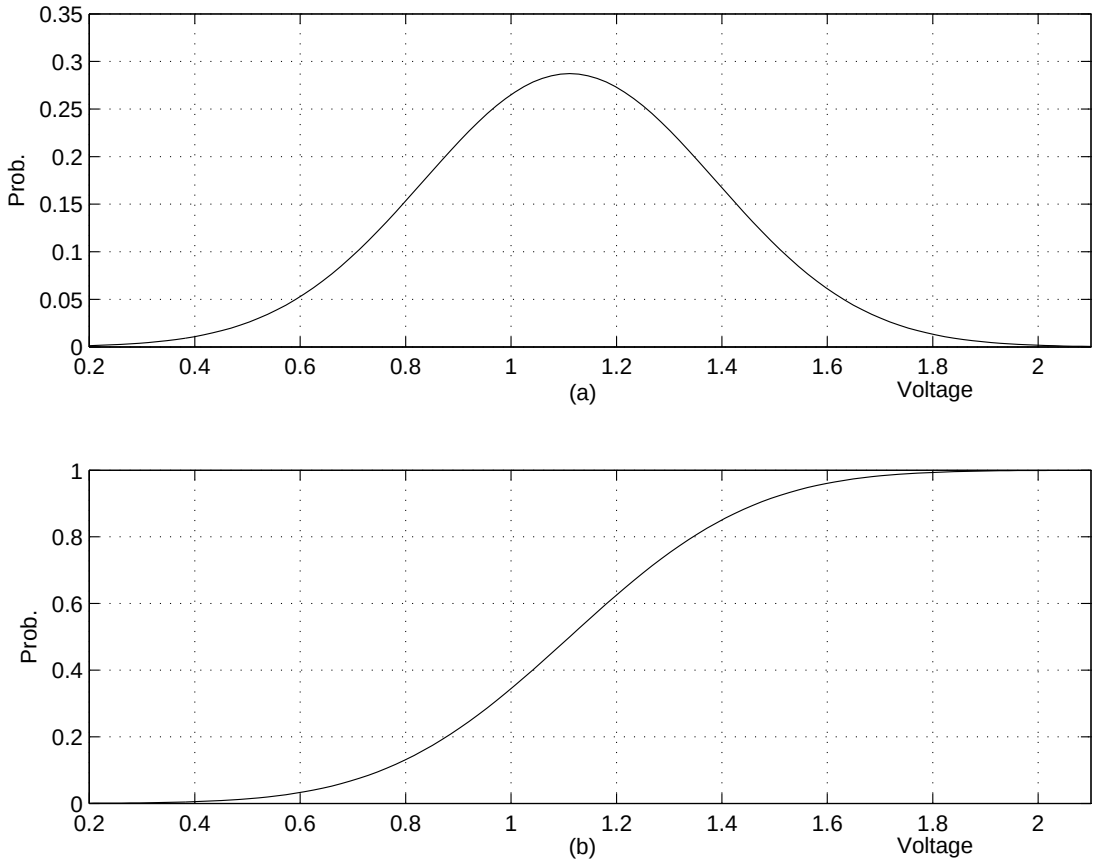


Fig. 7.10: Analytical method to estimate the distribution of voltage for 3σ=12% and η=79% (a) Density function of voltage. (b) Distribution function of voltage.

tribution function of voltage, respectively, for η = 89%, 3σ = 20%, and $b_r(opt)$ = 56-bit. The density function is normally distributed with the mean voltage of 0.89V. The results shown in Fig. 7.15 and 7.11 conclude that the analytically estimated mean voltage $\hat{\mu}_{V_{dd}}$ is high in case of η = 79% than the value of η = 89%. Fig. 7.12 shows the voltage distribution from the Monte Carlo simulation for 3σ=20% and η=79%. The simulation was carried out for 74169 iterations and the resulting CDF and PDF are normally distributed with the mean voltage of 1.14V as shown in Fig. 7.12(b). The analytical estimated mean voltage in Fig. 7.10 differs with the estimated mean voltage from the Monte Carlo simulation in Fig. 7.12.

The second part of experiments was conducted on the CMU Sphinx for speech recognition. We considered the speech lengths that varied from 1.06 to 11.8 sec and depending on the length of the speech, the recognition time changes. To shorten the recognition time, the FFT was configured to 256, 512, and 1024 points and burst size of 2, 4, and 8 for each configuration [193]. The data model of the communication task of the FFT $NB_{fft}(\zeta)$ was approximated as a normal distribution, with the mean $\mu_{NB_{fft}}$ = 248-bit and the standard deviation $\sigma_{NB_{fft}}$ = 44. However, the data model of the com-

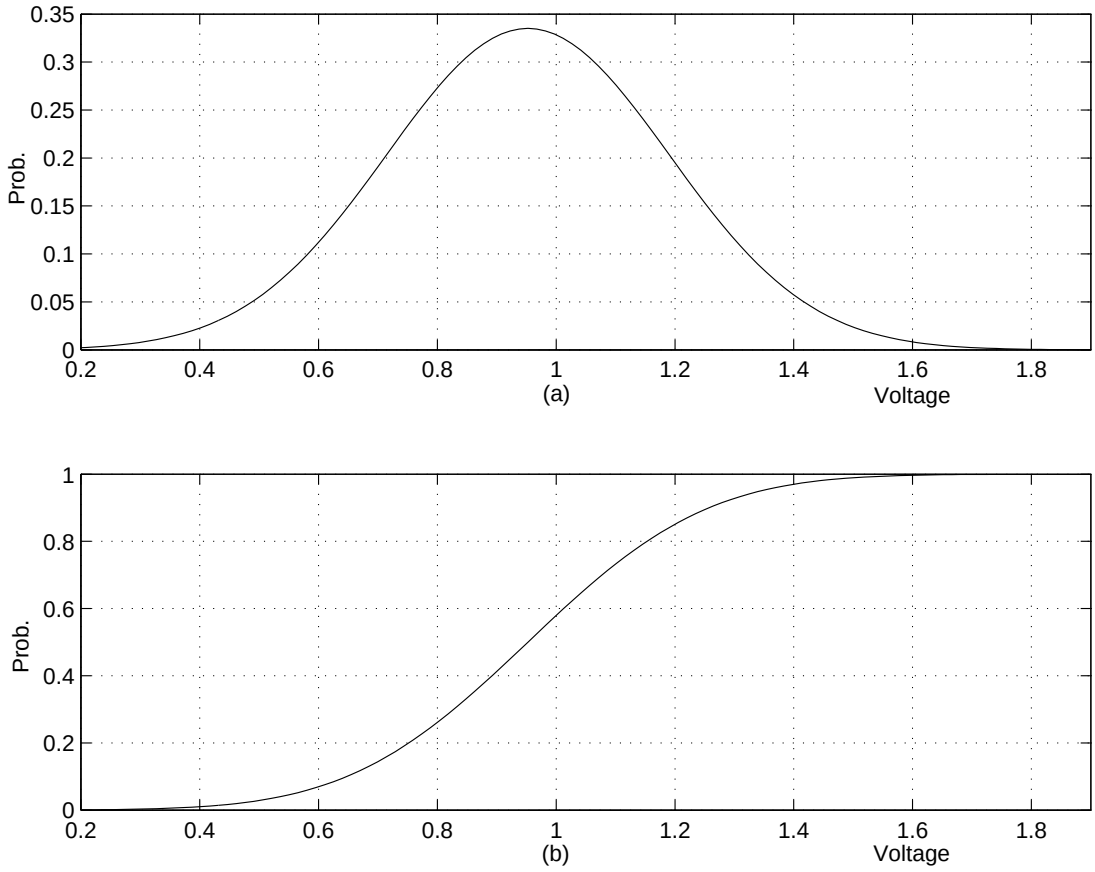


Fig. 7.11: Analytical method to estimate the distribution of voltage for 3σ=12% and η=89% (a) Density function of voltage. (b) Distribution function of voltage.

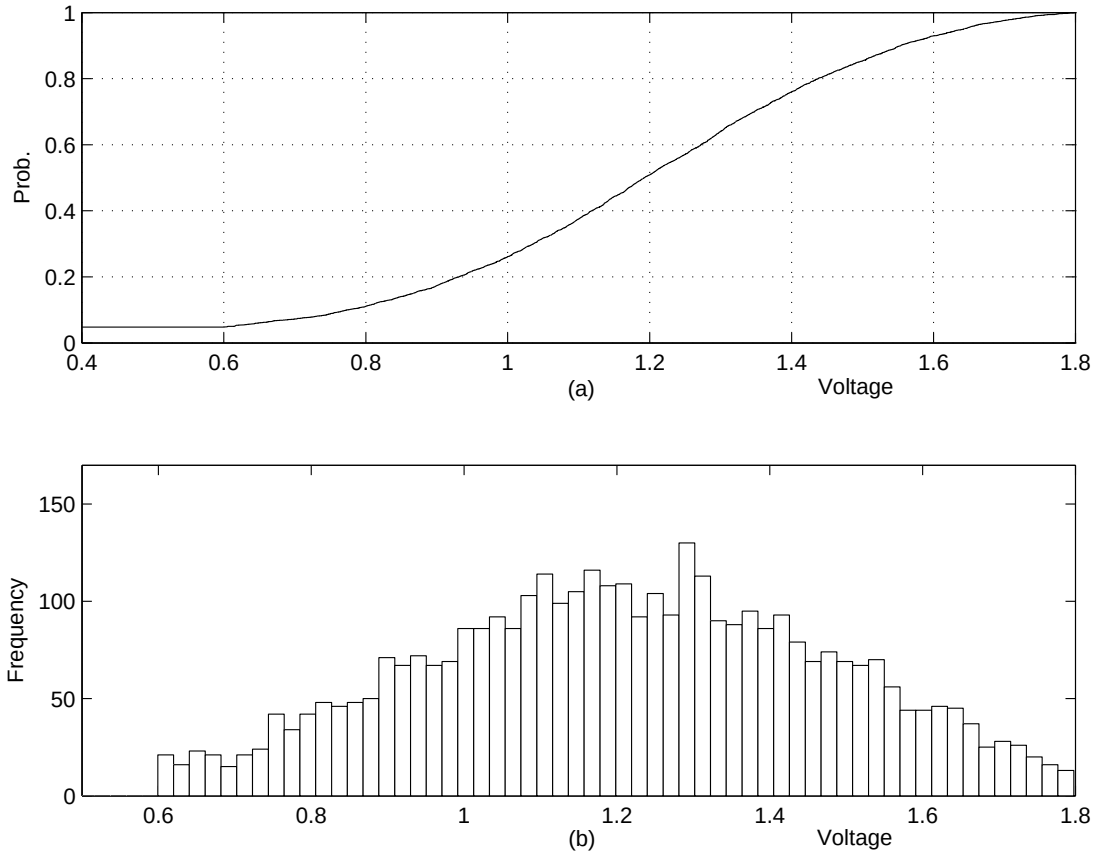


Fig. 7.12: Monte Carlo simulation to estimate the distribution of voltage for 3σ=12% and η=79% (a) Distribution function of voltage. (b) Density function of voltage.

Voltages	Frequency (Analytical)	Frequency (Monte Carlo)
0.6V	0.13	0.09
0.9V	0.42	0.34
1.2V	0.26	0.31
1.5V	0.15	0.19
1.8V	0.04	0.07

Tab. 7.6: Frequency of discrete voltages from analytical and Monte Carlo simulation with timing yield constraint η = 88%

munication tasks of a processor were kept deterministic. Fig. 7.13 shows the results of communication bus synthesis and the energy consumption for timing yield constraints ranging from 79% to 99%, using both analytical and Monte Carlo schemes. In this part of the experiment, we performed discrete voltage scaling with possible voltages V_{dd} = {0.6, 0.9, 1.2, 1.5, 1.8}. A constant single bus of 48-bit (cost [48/128]*100 = 37.5) was obtained for η ranging from 79% to 88% in Fig. 7.13(b), while the mean communication energy consumption was reduced upto 60% and 56% in Fig. 7.13(a) by scaling the voltage. For the timing yield constraint $\eta > 88\%$, the amount of slack is less, which offers less mobility of communication tasks c to share the same communication bus. Hence, for $\eta > 88\%$, two buses of 24 and 32-bit (cost 43.7) were obtained as shown in Fig. 7.13(b). Furthermore, in Fig. 7.13(a), it can be observed that the mean normalized energies (from analytical and Monte Carlo) are almost constant for the timing yield constraint $\eta \geq 96\%$. This is due to the fact that we used discrete voltage scaling techniques and at the higer value of timing yield constraint (in this case $\eta \geq 96\%$), a decrease in discrete voltage of an individual communication task is less likely to keep the minimum communication cost. Hence, in the above result of Fig. 7.13(a), reduction in mean normalized energies (from analytical and Monto Carlo) are almost constant for $\eta \geq 96\%$. In Tab. 7.6 the frequency of discrete voltages for timing yield constraint η = 88% are presented. The table compares the frequency of discrete voltages for the analytical and the Monte Carlo simulation methods. For example, in column 2 analytically estimated discrete voltages 1.2V, 1.5V, and 1.8V have a lower frequency than in column 3 using the Monte Carlo simulation. This results in a difference in energy consumption of the on-chip communication buses as shown in Fig. 7.13 (a).

7.3.2.3 Random Data Traffic and Process Varition

To validate the proposed bus synthesis technique under variations, we considered the randomly generated tasks and the real-life application speech recognition system. The randomly generated tasks consists of 119 communication tasks c and data to be transferred by all tasks c were assumed to be normally distributed with mean $\mu_c(NB)$ =

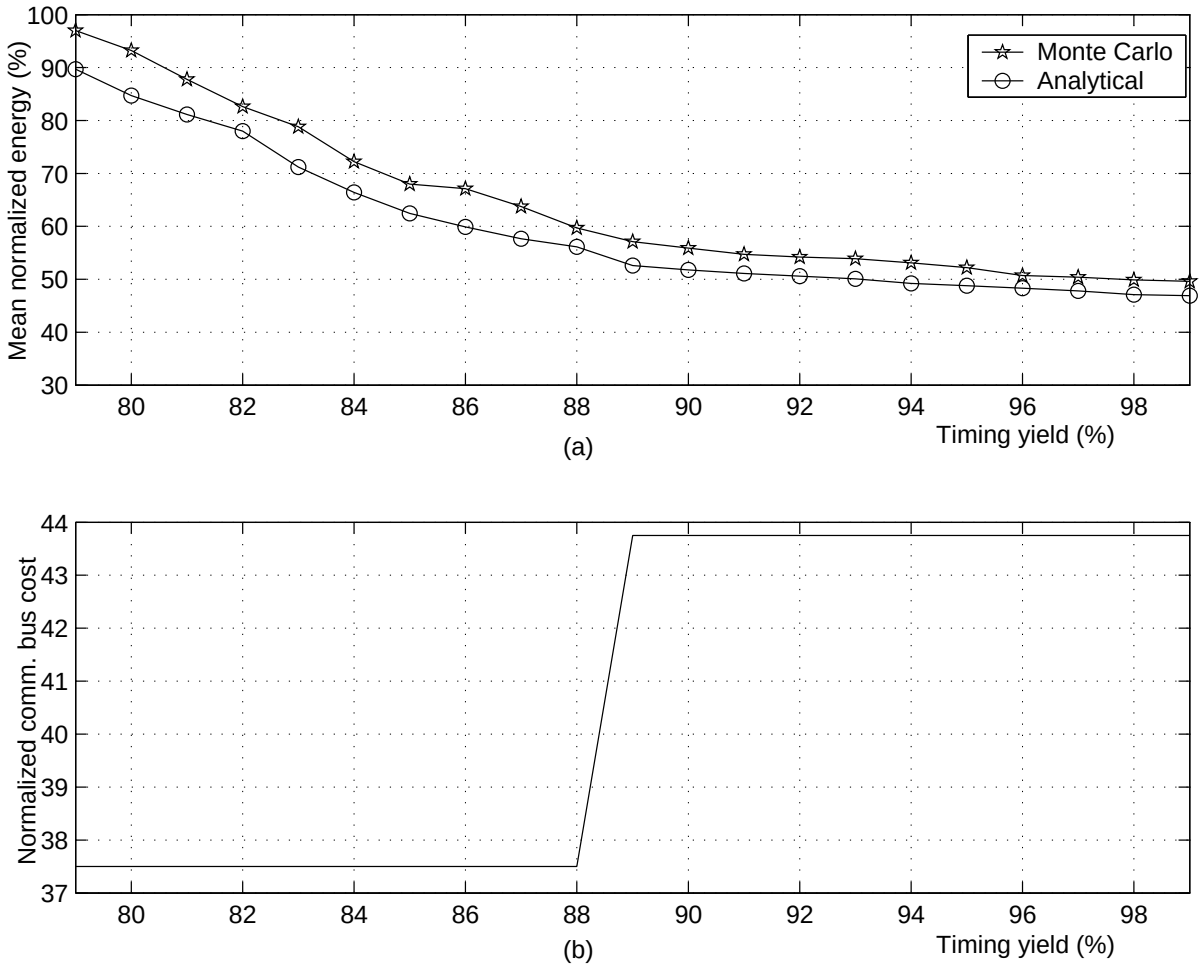


Fig. 7.13: Tuning of timing yield constraint (a) Mean normalized energy (b) Normalized communication bus cost for different timing yield constraints η.

512, 256, 128 and 64-bit. Different level of variability in data size $NB_c(\zeta)$ were explored ranging from 3% to 36% of $3\sigma_{NB}$. Similarly, an effect of process variations (V_{th}, L, T_{ox} and W) on the on-chip communication bus synthesis and voltage scaling, was also explored with a variation ranging from 2% to 12% of $3\sigma_{T_d}$. As assumed in [115, 36], these variation parameters are normally distributed and their values were extrapolated for 70nm CMOS technology from ITRS'05 [10] and the Berkeley predictive technology model [2] using the model presented in Eqs. (6.32) and (6.33). Furthermore, we assume that the random variables V_{th}, L, T_{ox}, and W are normally disributed. The data processing time w of each task τ are obtained from Eq. (4.1) for each communication task. We assume that discrete voltage pairs (supply and body bias voltages) of each communication task has been identified during the synthesis of on-chip modules. Further each on-chip module is capable to scale the voltages under the variation of load. Each communication task c can scale the supply voltage from 1.8V to 0.6V and the body bias voltage ranging 0V to -0.8V, to meet the desired timing yield constraint.

The first set of experiments was conducted to synthesize the optimal bus width and the number of buses with reduced communication energy under data size and process variations, using the voltage scaling technique (supply and body bias voltage). We performed the simultaneous scheduling, continuous voltage scaling, bus selection, and binding of communication tasks c using the proposed algorithm. Tab. 7.7 shows the results of optimized bus width and number of buses for the automatically generated tasks c, with change in $3\sigma_{T_d}$ due to process variation is 2%. The table compares the bus width and number of buses $b_r(opt)$, the mean supply voltage $\hat{\mu}_{V_{dd}}$, the mean body bias voltage $\hat{\mu}_{V_{bs}}$, and the mean slack $\hat{\mu}_{Slack}$ for the different timing yield constraint η. However, the confidence level α of all tasks $c \in C$, shown in Eq. (6.34) and (6.35) are fixed and set to 99.9% to meet the desired gate delay $T_{critical}$. The results of Tab. 7.7 show that the optimized bus width and number of buses change with the timing yield constraint η. In column 2 and 6 of the table, synthesized bus width $b_r(opt)$ are constant for two different timing yield constraints η = 79% and 99%, however, the mean voltages $\hat{\mu}_{V_{dd}}$ and $\hat{\mu}_{V_{bs}}$ decrease in column (3, 7) and (4, 8), respectively. In addition to this, the total mean slack $\hat{\mu}_{Slack}$ also decrease in column 5 and 9, respectively. This is because of a increase in the timing yield constraint of communication tasks from 79% to 89%. In column 10 of the table, there are two buses with different bus widths for all values of 3σ. In this case, timing yield constraint η of all the tasks are set to 99%, so that voltages of all the tasks c are scaled to the minimum possible value. This results in very small amounts of slack of communication tasks. Fig. 7.14 depicts the effect of process variations on the estimated supply voltage $\hat{\mu}_{V_{dd}}$ and body bias voltage $\hat{\mu}_{V_{bs}}$ for a fix $3\sigma_{NB}$ = 3%, $3\sigma_{T_d}$ = [2%, 4%, 6%, 8%, 10%, 12%] and timing yield constraint η = [79%, 89%, 99%]. The result shows that at a low value of timing yield constrait η, the effects of process variation is almost negligeable, i.e., the voltages are almost constant for all values of $3\sigma_{T_d}$. This is due to the fact that at a low value of timing yield

$3\sigma(NB)$	Timing yield η=79%				Timing yield η=89%				Timing yield η=99%				Run
	$b_r(opt)$ (bit)	$Ana.$ ($\hat{\mu}_{V_{dd}}$)	$Ana.$ ($\hat{\mu}_{V_{bs}}$)	$\hat{\mu}_{Slack}$ (%)	$b_r(opt)$ (bit)	$Ana.$ ($\hat{\mu}_{V_{dd}}$)	$Ana.$ ($\hat{\mu}_{V_{bs}}$)	$\hat{\mu}_{Slack}$ (%)	$b_r(opt)$ (bit)	$Ana.$ ($\hat{\mu}_{V_{dd}}$)	MC ($\hat{\mu}_{V_{bs}}$)	$\hat{\mu}_{Slack}$ (%)	time (sec.)
3σ=36%	64	0.89	-0.36	59.4	64	0.80	-0.40	41.4	(48,24)	0.76	-0.55	11.2	$\sim$ 86
3σ=33%	64	0.94	-0.35	58.2	64	0.79	-0.40	38.7	(48,24)	0.76	-0.55	9.5	$\sim$ 89
3σ=30%	60	0.99	-0.37	57.9	60	0.80	-0.40	38.3	(36,32)	0.75	-0.54	9.7	$\sim$ 97
3σ=27%	60	1.02	-0.35	57.3	60	0.85	-0.39	35.9	(36,32)	0.75	-0.52	9.3	$\sim$ 107
3σ=24%	56	1.04	-0.36	55.8	56	0.86	-0.39	35.6	(32,32)	0.77	-0.52	9.8	$\sim$ 124
3σ=21%	56	1.09	-0.34	54.4	56	0.86	-0.36	34.5	(32,32)	0.79	-0.49	7.9	$\sim$ 139
3σ=18%	48	1.16	-0.35	54.8	48	0.87	-0.36	34.8	(32,32)	0.81	-0.46	7.0	$\sim$ 146
3σ=15%	48	1.15	-0.34	51.1	48	1.06	-0.34	32.6	(32,32)	0.87	-0.43	7.1	$\sim$ 157
3σ=12%	36	1.25	-0.32	49.3	36	1.07	-0.34	32.9	(32,16)	0.93	-0.44	6.3	$\sim$ 169
3σ=9%	36	1.26	-0.32	49.7	36	1.14	-0.35	31.1	(32,16)	0.94	-0.42	6.4	$\sim$ 177
3σ=6%	32	1.27	-0.30	48.6	32	1.07	-0.31	31.4	(32,16)	1.03	-0.41	6.5	$\sim$ 186
3σ=3%	32	1.27	-0.26	48.1	32	1.21	-0.31	30.2	(32,16)	1.06	-0.39	6.6	$\sim$ 198

Tab. 7.7: Synthesize buses and supply/body bias voltages for different timing yield constraint (η), standard deviation ($3\sigma_{NB}$) of data size and $3\sigma_{T_d}$ = 2%

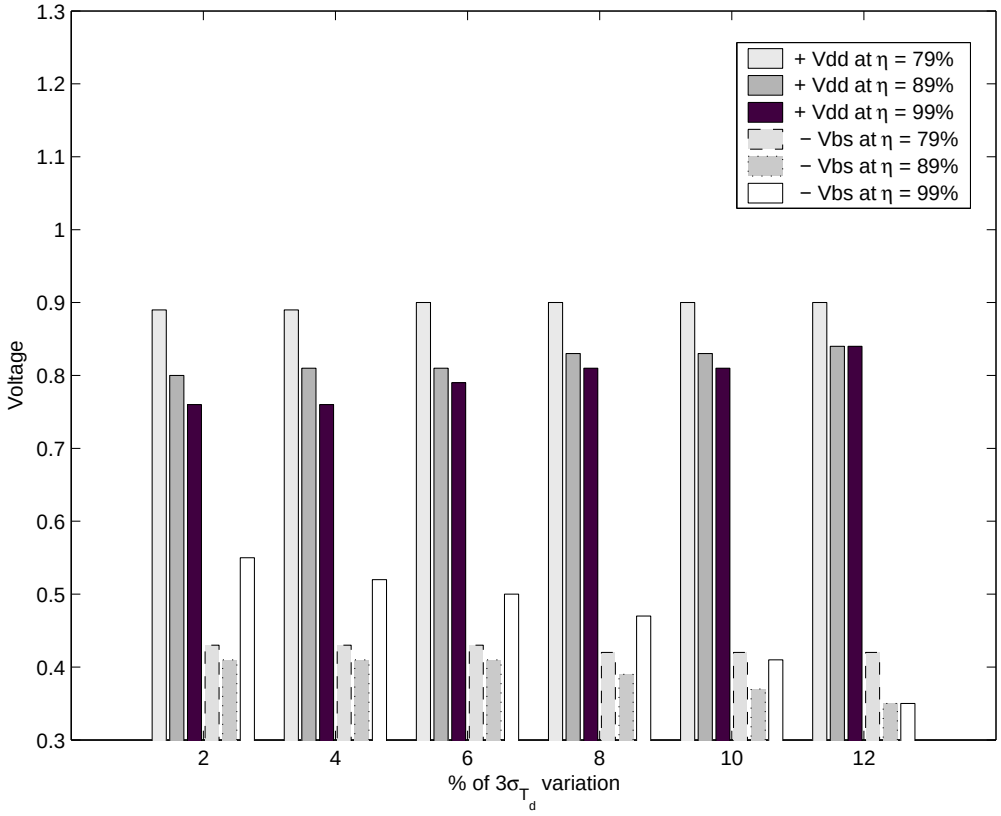


Fig. 7.14: Supply and body bias voltage for $3\sigma_{NB} = 3\%$, different timing yield constraints, process variations

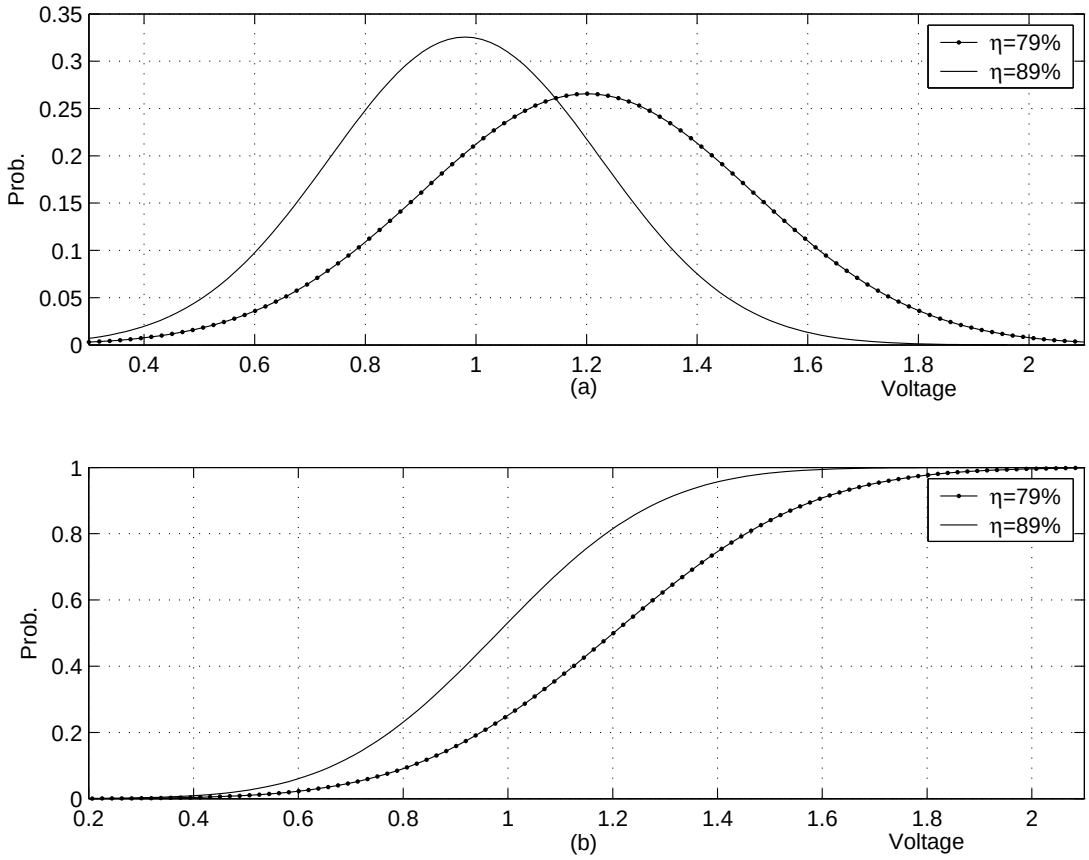


Fig. 7.15: Analytical method to estimate the voltage distribution for $3\sigma_{NB}$=18%, η=79% and $3\sigma_{T_d}$ = 2% (a) Density function of voltage. (b) Distribution function of voltage.

constraint η, the amount of slack is high as shown in Tab. 7.7, and this slack can be used to compensate the effect of process variations. However, at a high value of timing yield constraint, the amount of slack is lower as shown in Tab. 7.7 and this slack is not an enough to fully compensate the effects of process variations. This results in an increase of supply voltage $+\hat{\mu}_{V_{dd}}$ and body bias voltage $-\hat{\mu}_{V_{bs}}$ as shown in Fig. 7.14. The ratio of increase in $+\hat{\mu}_{V_{dd}}$ is less than the ratio of increase in $-\hat{\mu}_{V_{bs}}$. This is because the body bias voltage is scaled at first in presence of process variations and the supply voltage is scaled only when T_d does not meet the constraint $T_{critical}$ as shown in Eq. (6.35). Fig. 7.15 depicts, the analytical estimation of the probability density function and the cumulative distribution function of supply voltage for η = (79%, 89%), $3\sigma_{NB}$ = 18%, $b_r(opt)$ = 48-bit, and process variation $3\sigma_{T_d}$ = 2%. Fig. 7.15(a) shows the density function of voltages with a mean of 1.11V and 0.87V for η = 79% and 89%, respectively. Their corresponding CDFs are shown in Fig. 7.15(b) for voltage range V_{dd} = 0 to 2.5V. The results shown in Fig. 7.15 conclude that the analytically estimated mean of supply voltage $\hat{\mu}_{V_{dd}}$ is higher in case of η = 79% than for a value of η = 89%.

In Fig. 7.16, the estimated distribution of supply voltage is shown using the Monte

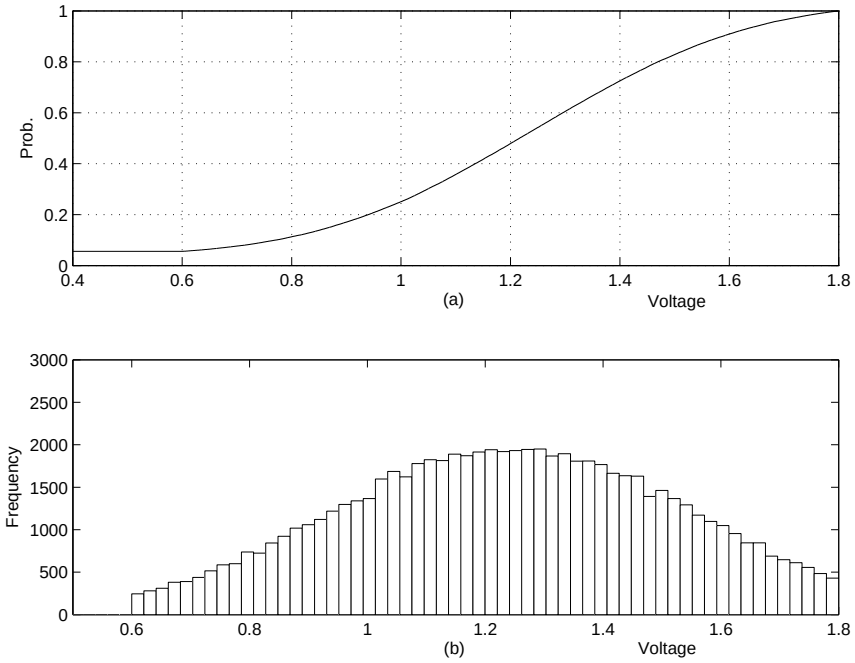


Fig. 7.16: Monte Carlo simulation to estimate the distribution of voltage for $3\sigma_{NB}$=18%, η=79% and $3\sigma_{T_d}$ = 2% (a) Distribution function of voltage. (b) Density function of voltage.

Carlo simulation for η = 79%, $3\sigma_{NB}$ = 18%, $b_r(opt)$ = 48-bit and process variation $3\sigma_{T_d}$ = 2%. The simulation was carried out for 74169 iterations and the resulting shape of CDF and PDF are normal in Fig. 7.16(a) and (b), respectively, with a mean supply voltage of 1.07V as shown in Fig. 7.12(b).

The second part of experiment was conducted on the speech recognition system. The FFT was configured to 256, 512 and 1024 point and burst size of 3 to 6. The data model of the FFT $NB_{fft}(\zeta)$ was approximated as a normal distribution, with the mean $\mu_{NB_{fft}}$ = 248-bit and the standard deviation $\sigma_{NB_{fft}}$ = 44. While, the data model of the communication tasks of a processor was kept deterministic. Fig. 7.17 shows the results of communication bus synthesis and mean energy consumption for timing yield ranging from 79% to 99% and the process variations $3\sigma_{T_d}$ ranging from 2% to 12%. In this part of the experiment, we performed discrete voltage scaling with discrete supply voltages {0.6V, 1.0V, 1.2V, 1,4V, 1.6V} and body bias voltages {0V, -0.2V, -0.4V, -0.6V, -0.8V}. A constant single bus of 48-bit with a cost of 37.5 (communication bus cost is normalized to a 128-bit bus) was obtained for η ranging from 79% to 88% in Fig. 7.17(b), while the mean communication energy consumption was reduced from 63% to 56% for a different percentages of $3\sigma_{T_d}$ in Fig. 7.17(a), due to the supply and body bias voltage scaling. For the timing yield $\eta > 88\%$, the amount of slack is less. Hence, for the $\eta > 88\%$, two buses of 24 and 32-bit (cost (24+32)/120 = 43.7) was obtained as shown in Fig. 7.17(b). This communication bus cost is constant up to the timing

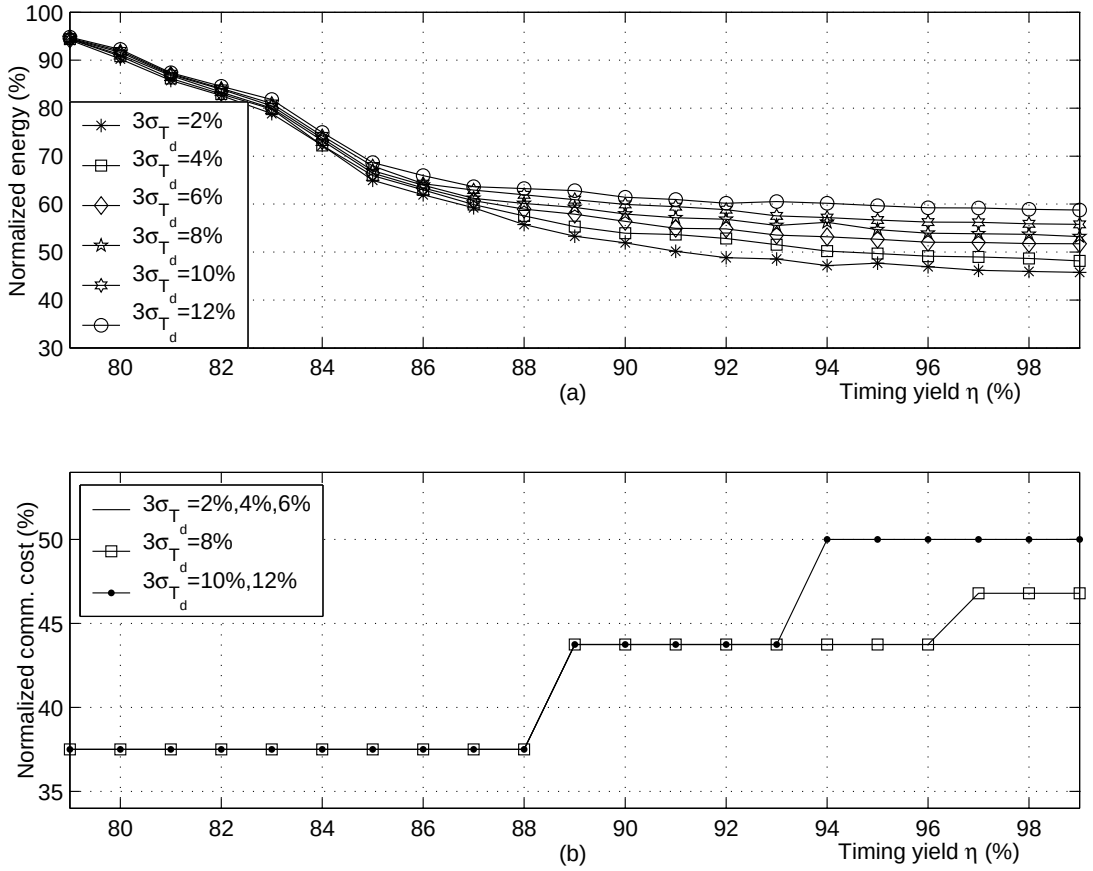


Fig. 7.17: Tunning of timing yield constraint (a) Mean normalized energy consumption (b) Normalized communication bus cost as a function of timing yield η.

yield constraint η = 99% for the process variations $3\sigma_{T_d}$ = 2%, 4% and 6%. While the mean communication energy consumption was decreased further from 52% to 46%. In Fig. 7.17(b) communication cost was increased from 43.7% (two 32-bit bus) to 50% for process variation $3\sigma_{T_d}$ = 10% and 12% at the timing yield $\eta \geq 94\%$. Similary, the communication bus cost was increased from 43.7% to 46.8% (two bues of 28 and 32-bit) for process variations $3\sigma_{T_d}$ = 8% and the timing yield $\eta \geq 97\%$.

7.4 Summary

Summarizing the experiments, a trade-off between the minimization of buses and its energy reduction was explored by varying the timing yield constraint during the synthesis of on-chip communication buses. We have noticed that increasing the timing yield constraint η can reduce the energy consumption, however, if the value of η increases from a certain limit, the mobility of communication tasks will reduced and results in the use of additional bus resources (increase in bus width or increase in the

number of buses) to meet the real-time constraints. Therefore, the timing yield constraint η can be used as a tuning factor to synthesize the optimal bus width and number of buses with the reduced communication energy consumption. Furthermore, we observed that estimating the statistical parameters of voltage using an analytical method is quite close to the results of Monte Carlo simulation.

Chapter 8

Conclusion and Future Work

Contents

In this thesis, we have presented an approach to synthesize the bus widths and the number of buses, while at the same time scaling the bus voltages, all in an effort to reduce bus costs and energy consumption taking into account communication traffic and process variations. The synthesis problem is formulated into three main steps: scheduling, allocation, and binding problems, assuming that a system has been partitioned and mapped onto the appropriate modules of an SoC. In order to find a trade-off between bus costs and energy consumption, simultaneous bus synthesis and voltage techniques have been employed. The results show that the proposed technique synthesizes an energy efficient communication bus that results in an effective utilization of buses under data size and process variations.

8.1 Contributions

The main contribution of this work is to synthesize the bus widths and the number of buses for MPSoC architectures. In **Chap.** 2 several factors ranging from technology to architectures have been discussed that influence the performance of on-chip communication buses. Although semiconductor technology is playing an important role to be able to integrate several functionalities on a single chip, communication can become critical as the number of processing units increase on an SoC. Further, it has been shown that it is equally essential to optimize the communication bus architectures in terms of power and performance at each level of abstraction. Taking into account the

challenges caused by increasing system complexity and technology scaling, communication bus synthesis algorithms under real-time constraints are presented in **Chap.** 4. At first the bus synthesis problem is formulated in a mathematical programming and scheduled communication tasks for different bus widths in order to find the minimum number of overlaps among them. While this formulation finds the optimal solution, its run time complexity is **NP**-hard, which means it is unusable for a system with a large problem size. Thus a heuristic is proposed based on tabu search, which finds a near-optimal solution in polynomial time complexity. In order to find the number of buses and the topology, a clique partitioning algorithm (Algorithm 4.7) is used. Furthermore, a communication bus architecture refinement technique is presented to increase the *locality of communication* such that bridges interconnecting buses are used rarely. This results in a further reduction of power and delay overhead. The results of Tab. 4.1 and 4.5 show that even after scheduling communication tasks using the above algorithms, there may be still significant amounts of slack left and result in the underutilization of the on-chip communication buses, which mean that the on-chip communication buses remain underutilized.

In order to use the buses more effectively, in **Chap.** 5 simultaneous on-chip communication bus synthesis and voltage scaling technique are described. The technique exploits the slack for both bus sharing and voltage scaling in order to find the minimum communication cost with reduced energy consumption. At first a combined bus synthesis and supply voltage scaling technique is presented with the aim to minimize communication cost (Eq. (5.11)) subject to a set of constraints (Eq. (5.12) - Eq. (5.19)). The supply voltage scaling technique reduces the bus energy consumption, however, the voltage cannot be scaled to the minimum level because of signal integrity problems. Thus, we presented an extended energy aware bus synthesis model based on both supply and body bias voltages scaling. The power and delay models (Eq. (5.21) - Eq. (5.24)) of communication buses show that the body bias voltage level has a higher impact to power consumption and less on delay in comparison to the supply voltage level. Therefore, in the extended bus synthesis model, the body bias voltage is scaled to the minimum level in order to reduce the leakage power consumption, however, the supply voltage is scaled to exploit the rest of the slack left after body bias voltage scaling. As the continuous voltage scaling problem can be solved in polynomial time complexity and results in a better energy consumption characteristic, it cannot be employed for the digital design due to its implementation costs. Thus a discrete voltage scaling technique is used, which is, however, known to be **NP**-hard. To master the complexity of this discrete voltage scaling technique, heuristics are proposed (Algorithm 5.1 and 5.2) for both supply voltage scaling and supply and body bias voltages scaling, respectively. The heuristics are based on a linear relaxation method, which solves the problem in a *quasi-polynomial* time complexity. The bus synthesis algorithms presented in **Chap.** 4 and 5 assume that the amount of data to be transferred between

communication tasks is fixed, however, this does not apply to a system with a variable workload. Further, in both chapters, an α power delay model is considered to model the gate delay. The model is deterministic and does not take into account the effects of process variations on the performance of on-chip communication buses.

To incorporate the effects of data size and process variations on the performance of communication buses a rigorous bus synthesis and voltage scaling models are presented in **Chap.** 6. At first the problem of combined bus synthesis and supply voltage scaling under data size variation is formulated. In this model, the data transfer delay CLTI of each communication task is described as a function of random data size (Eq. (6.8)) and its time constraint is modeled as a probabilistic constraint (Eq. (6.9)). Later the probabilistic constraint is relaxed to a deterministic nonlinear constraint for each communication task (Eq. (6.11)). The bus synthesis and voltage scaling problem with a nonlinear constraint is an optimization problem, which finds a trade-off between communication bus cost and energy consumption for a given timing yield constraint. An algorithm (Algorithm 6.2) is presented to find the best timing yield constraint for the minimum communication bus cost and energy consumption. Further an analytical model of the voltage density function (Eq (6.25)) is derived to estimate the values of supply voltage under variable data sizes. Second an extended bus synthesis and voltage scaling model is presented in order to cope with the variability in the process parameters. It combines the effects of both data size and process variations and synthesizes energy aware robust on-chip communication bus architectures. Instead of the α power delay model, a rigorous gate delay model (Eq. (6.27)) with short channel effects and process variations are considered. The resulting synthesis problem is relaxed to a convex quadratic optimization problem to minimize the communication bus cost (Eq. (5.11)) with a set of constraints (Eq. (6.41) and (6.42)) for both process and data size variations, respectively.

8.2 Possible Future Work

In this thesis algorithms and techniques to synthesize energy conscious on-chip communication buses are presented. Beyond this, there is a wide array of research possibilities and challenges that can be solved by extending the proposed synthesis model. Some of these are bus synthesis for dynamic on-chip data traffic, communication protocol synthesis, power and thermal effect modeling and optimization using different techniques, and simultaneous bus synthesis and retiming.

Bus synthesis for dynamic on-chip data traffic: In this thesis communication activities between on-chip communicating modules and their data size are extracted by profiling several applications statically. Recently, several research works in reconfig-

urable computing have shown that reconfigurable architectures can meet dynamically the computation demanded by different applications. This, in turn, results in on-chip data traffic that cannot be modeled by using a static profiling approach. Thus a proper modeling technique is needed to model this kind of data traffic and then our optimization technique could be used to synthesize the communication buses.

Communication protocol synthesis: As discussed in **Chap.** 2 that the communication protocols have a significant impact on the performance of bus architectures. In this thesis, we intended to synthesize communication protocols after the synthesis of communication buses. However, different standard protocols provided by vendors such as AMBA bus, CoreConnect, etc. can be used at early bus synthesis phase in order to explore different protocols.

Power and thermal effect modeling and optimization using different techniques: There are several optimization techniques to reduce the power consumption of on-chip communication buses. These include voltage scaling, bus encoding, bus splitting, frequency scaling, and changing the duty cycle of clock. However, among them, we only used the voltage scaling technique in order to exploit the dynamic slack. A possible future work would be to extend this work to integrate and to explore different existing power optimization techniques. Furthermore, a recent data shows that more than 50% of all integrated circuit failures are related to thermal issues. Thus, the design of future nanometer chips requires the accurate and simultaneous modeling, estimation, and optimization of power and thermal effects at a high level of abstaction.

Simultaneous bus synthesis and retiming: In the era of about a billion transistors on a single chip, signals cannot reach across the chip within a cycle. The estimated results show that less than 1% of a chip will be reachable in a single clock cycle [13]. In the past, several methods including repeater based approach has been used to reduce the wire delay, however, this technique cannot be employed completely to enhance the required delay. An interesting direction would be to insert different synchronous storage stages (registers) between two ends of a bus such that the data will be transferred in a pipeline fashion. This problem is a well known retiming problem, which can be easily integrated into the bus synthesis and optimization technique presented in this thesis.

Appendix A

Mathematical Programming

Mathematical programs are the most widely used models to optimize the different problems in the area of operation research, electrical engineering, control engineering etc. The main objective is to find a global optimum solution of a function $f(x_1, x_2, \cdots, x_n)$ with respect to a set of m constraints $h_j(x_1, x_2, \cdots, x_m) \leq c_j$ $(j = 1, \cdots, m)$ and bounds for the n variables $(lb_i \leq x_i \leq ub_i,\ i = 1, \ldots, n)$. If objective function and a set of constraints are linear function of the variables then the problem is called the linear programming. If all the variables are defined as a binary integer variable then the problem is called an integer linear programming (ILP) [142] and if some of them are binary integer variable then the problem of this class is called a mixed integer linear programming (MILP). In addition to this, if either f or h_j are nonlinear functions, this type of problem is called a nonlinear programming (NLP). If some of the constraint h_j or the objective function f are expressed in terms of probabilistic statement, the problem is called a stochastic nonlinear programming. In general solving the problem of stochastic NLP is known to be **NP**-hard. However, in [117] several efficient convex nonlinear optimization algorithms are proposed that optimize the problem in a polynomial time complexity.

Appendix B

Convex Functions

Contents

Convex optimization is a branch of mathematics dealing with a nonlinear programming problems with additional geometric structure. This area has been the focus of considerable research due to the fact that convex optimization problems are scalable and can be efficiently solved by interior-point methods. Additionally, convex optimization problems are much more prevalent than previously thought as existing problems are constantly being recast in a convex framework. There exists a bunch of convex commercial and non-commercial solvers such as MOSEK [5], NEOS (Network Enabled Optimization System) [6], Cplex [3]. Among them MOSEK and Cplex are the commercial solvers. While the NEOS is the non-commercial solver, which solve an optimization problem remotely over the internet. However, there is no guarantee as to the schedule or volume of computing resources to be made available. The CPLEX is a commercial solver, which is developed to solve large, difficult problems where other linear programming solvers fail or are unacceptably slow. Another commercial solver MOSEK solves a large-scale linear, convex quadratic, conic quadratic, and smooth convex optimization problems of unlimited size (limited by computer memory only). It handles integer variables for linear, quadratic, and convex quadratically constrained optimization problems. It consists of three different optimizers: interrior-point, primal simplex, and mixed integer. In this work, we used MOSEK solver to solve a convex optimization problem with a limited number of variables and constraints. However, it was sufficient to validate the methodology.

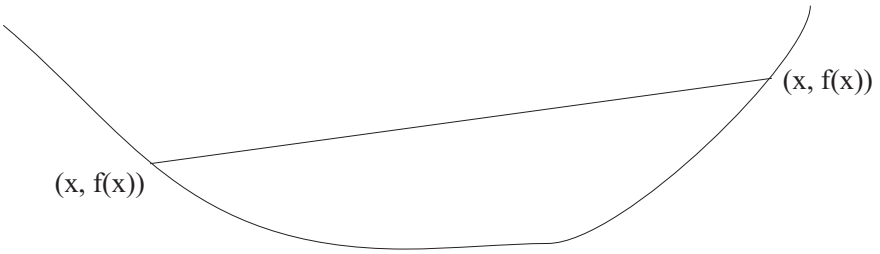


Fig. B.1: A convex function. The chord between any two points on the graph lies above the curve [30]

B.1 Definition

A function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is *convex* if **dom** f is a convex set and if for all $x, y \in$ **dom**f, and θ with $0 \leq \theta \leq 1$, we have [30]

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y) \tag{B.1}$$

Geometrically, this inequality means that the line segment between $(x, f(x))$ and $(y, f(y))$, which is the *chord* from x to y, lies above the graph of f as shown in Fig. B.1. A function f is *strictly convex* it strict inequality holds in Eq. (B.1) whenever $x \neq y$ and $0 < \theta < 1$. Further f is *concave* if $-f$ is convex, and *strictly concave* if $-f$ is strictly convex. For an affine function there is always equality in Eq. (B.1), so all affine (and therefore also linear) functions are both convex and concave. Conversely, any function that is convex and concave is affine. A function is convex if and only if it is convex when restricted to any line that intersects its domain. In other words f is convex if and only if for all $x \in$ **dom** f and all v, the function $g(t) = f(x + tv)$ is convex (on its domain, $\{t|x + tv \in$ **dom**$f\}$). This property is very useful, since it allows us to check whether a function is convex by restricting it to a line. More about convex optimization can be found in [30].

B.2 First Order Conditions

Suppose f is differentiable (i.e., its gradient ∇f exists at each point in **dom** f, which is open). Then f is convex if and only if **dom** f is convex and

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{B.2}$$

holds for all $x, y \in$ **dom** f. The inequality in Eq. (B.2) shows that from *local information* about a convex function (i.e., its value and derivative at a point) we can derive *global information*. This is perhaps the most important property of convex functions, and explains some of the remarkable properties of convex functions and convex optimization

problem. As one simple example, inequality of Eq. (B.2) shows that if $\nabla f(x) = 0$, then for all $y \in$ **dom** f, $f(y) \geq f(x)$, i.e., x is a global minimizer of the function f. Strict convexity can also be characterized by a first-order condition: f is strictly convex if and only if **dom** f is convex and for $x, y \in$ **dom** f, $x \neq y$. For concave functions we have the corresponding characterization: f is concave if and if **dom** f is convex and

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) \tag{B.3}$$

for all $x, y \in$ **dom** f.

B.3 Second Order Conditions

First assume that a function f is twice differentiable, that is, its *Hessian* or second derivative $\nabla^2 f$ exists at each point in **dom** f, which is open. Then f is convex if and only if **dom** f is convex and its Hessian is positive semidefinite: for all $x \in$ **dom** f,

$$\nabla^2 f(x) \geq 0 \tag{B.4}$$

For a function on **R**, this reduces to the simple condition $f''(x) \geq 0$ (and **dom** f convex, i.e., an interval), which means that the derivative is nondecreasing. The condition $\nabla^2 f(x) \geq 0$ can be interpreted geometrically as the requirement that the graph of the function have positive (upward) curvature at x.

Similarly, f is concave if and only if **dom** f is convex and $\nabla^2 f(x) \leq 0$ for all $x \in$ **dom** f. Strict convexity can be partially characterized by second order conditions. If $\nabla^2 f(x) > 0$ for all $x \in$ **dom** f, then f is strictly convex. The converse, however, is not true.

Appendix C

Technology Parameters

This appendix enumerates briefly the relevent technological parameters for CMOS 70nm technology. More information can be found in Berkeley predictive technology model [2].

Variable	Values	Unit
K_1	0.063	-
K_2	0.153	-
K_3	5.38e-07	-
K_4	1.83	-
K_5	4.19	-
K_6	5.26e-12	-
K_7	-0.144	-
C_{eff}	2.0e-15	F
I_j	4.8e-10	A
$V_{th}0$	0.423	V
V_{dd}	1.0	V

Tab. C.1: Technology dependent parameters

References

[1] AMBA 2.0 Specification. www.arm.com/products/solutions/AMBAOverview.html.

[2] Berkeley Predective Technology Model. www-device.eecs.berkeley.edu.

[3] Cplex optimizier. www.cplex.com.

[4] IBM CoreConnect. www.chips.ibm.com/products/powerpc/cores.

[5] MOSEK Optimization Software. www.mosek.com/documentation.html♯manuals.

[6] Neos solver. http://www-neos.mcs.anl.gov/.

[7] Open Core Protocol International Partnership (OCP-IP). www.ocpip.org.

[8] The CMU sphinx group open source speech recognition engines. www.speech.cs.cmu.edu/sphinx/.

[9] Vorbis I Specification. http://www.xiph.org/ogg/vorbis/doc/VorbisIspec.html.

[10] International Technology Roadmap for Semiconductors. http://public/itrs/net., 2005.

[11] S. ABDI, D. SHIN, and D. GAJSKI. Automatic Communication Refinement for System Level Design. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2003.

[12] S. N. ADYA and I. L. MARKOV. Fixed-Outline Floorplanning: Enabling Hierarchical Design. *IEEE Trans. Very Large Scale Integrated (VLSI) Systems*, 2003.

[13] V. AGARWAL, M. HRISHIKESH, S. W. KECKLER, and D. BURGER. Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures. In *proc. of Int. Symposium on Computer Architecture (ISCA)*, 2000.

[14] G. AGOSTA, F. BRUSCHI, and D. SCIUTO. Static Analysis of Transaction-Level Models. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2003.

[15] C. J. ALPERT, A. DEVGAN, and S. T. QUAY. Buffer Insertion with Accurate Gate and Interconnect Delay Computation. In *proc. of Design Automation Conference (DAC), New Orleans, Louisiana*, 1999.

[16] A. ANDREI, M. SCHMITZ, P. ELES, Z. PENG, and B. AL-HASHIMI. Overhead Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time Constrained Systems. In *proc. of Design Automation and Test Europe (DATE)*, 2004.

[17] A. ANDREI, M. SCHMITZ, P. ELES, Z. PENG, and B. M. A. HASHIMI. Simultaneous Communication and Processor Voltage Scaling for Dynamic and Leakage Energy Reduction in Time Constrained Systems. In *proc. of Int. Conf. on Computer-Aided Design (ICCAD)*, 2004.

[18] F. BELINA, D. HOGREFE, and A. SARMA. *SDL with Applications from Protocol Specifications*. Carl Hanser Verlag and Prentice Hall International (UK) Ltd., 1991.

[19] L. BENINI, A. BOGLIOLO, and G. D. MICHIELI. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, 2000.

[20] L. BENINI, A. MACII, E. MACCI, M. PONCINO, and R. SCARSI. Architectures and Synthesis Algorithms for Power-Efficient Bus Interfaces. *IEEE Trans. on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 19(9):969–980, 2000.

[21] L. BENINI and G. D. MICHELI. *Dynamic Power Management : Design Techniques and CAD Tools*. Kluwer Academic Publishers, 1998.

[22] L. BENINI and G. D. MICHELI. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35:70–78, 2002.

[23] L. BENINI, G. D. MICHELI, and E. MACII. Designing Low-Power Circuits: Practical Recipes. *IEEE Circuits and Systems Magazine*, 1(1):6–25, 2001.

[24] L. BENINI, G. D. MICHELI, E. MACII, D. SCIUTO, and C. SILVANO. Address Bus Encoding Techniques for System Level Power Optimization. In *proc. of Design Automation and Test Europe (DATE)*, 1998.

[25] J. R. BIRGE and F. LOUVEAUX. *Introduction to Stochastic Programming*. Springer Series in Operation Research, 1997.

[26] M. BOLT, M. ROCCHI, and J. ENGEL. Realistic Statistical Worst-Case Simulation of VLSI Circuits. *In IEEE Transactions on semiconductor manufacturing*, Vol. 4(No. 3):193–198, 1991.

[27] S. BORKAR, T. KARNIK, S. NARENDRA, J. TSCHANZ, A. KESHAVARZI, and V. DE. Parameter Variations and Impact on Circuits and Microarchitecture. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2003.

[28] D. S. BORMANN and P. Y. K. CHEUNG. Asynchronous Wrapper for Heterogenous Systems. In *proc. of Int. Conf. Computer Design (ICCD)*, 1997.

[29] K. A. BOWMAN, B. L. AUSTIN, J. C. EBLE, X. TANG, and J. D. MEINDL. A Physical Alpha-Power Law MOSFET Model. *IEEE Journal of Solid-State Circuits*, 34(10):1410–1414, 1999.

[30] S. P. BOYD and L. VANDENBERGHE. *Convex Optimization*. Cambridge University Press - Publisher, 2004.

[31] J.-Y. BRUNEL, E. A. KOCK, W. M. KRUIJTZER, K. J. H. N. KENTER, and W. J. M. SMITS. Communication Refinement in Video Systems on Chip. In *proc. of Int. Workshop on Hardware/Software Co-design*, 1999.

[32] J.-Y. BRUNEL, W. KRUIJTZER, H. KENTER, F. PETROT, and L. PASQUIER. COSY Communication IPs. In *proc. Design Automation Conference (DAC)*, 2000.

[33] J. BUCK, S. HA, E. A. LEE, and D. G. MESSERSCHMITT. PTOLEMY: A Framework for Simulating and Prototyping Heterogeneous Systems. *Int. Journal on Computer Simulation*, pages 1–34, 1992.

[34] J. T. BUCK. Static Scheduling and Code Generation from Dynamic Dataflow Graphs with Integer Valued Control Streams. In *proc. of Int. Conf. on Signals, Systems, and Computers*, 1994.

[35] T. D. BURD, T. A. PERING, A. J. STRATAKOS, and R. W. BRODERSEN. A Dynamic Voltage Scaled Microprocessor System. *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580, 2000.

[36] Y. CAO and L. T. CLARK. Mapping Statistical Process Variations Toward Circuit Performance Variability: An Analytical Modeling Approach. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2005.

[37] J.-M. CHANG and M. PEDRAM. Energy Minimization Using Multiple Supply Voltages. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 5(4):436–443, 1997.

[38] C. CHEN and M. SARRAFZADEH. Power reduction by simultaneous voltage scaling and gate sizing. In *proc. of Asia South Pacific Design Automation Conf. (ASPDAC)*, 2000.

[39] C. CHEN, A. SRIVASTAVA, and M. SARRAFZADEH. On Gate Level Power Optimization Using Dual-Supply Voltages. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 9(5):616–629, 2001.

[40] T. CHEN and S. NAFFZIGER. Comparision of Adaptive Body Bias (ABB) and Adaptive Supply Voltage (ASV) for Improving Delay and Leakage Under the Presence of Process Variation. *IEEE transactions on very large scale integration (VLSI) systems*, 11(5):888–899, 2003.

[41] J. CONG, Y. FAN, G. HAN, X. YANG, and Z. ZHANG. Architecture and Synthesis for On-Chip Multicycle Communication. *IEEE Trans. on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 23(4):550–564, 2004.

[42] J. CONG and K. S. LEUNG. Optimal Wiresizing Under the Distributed Elmore Delay Model. In *proc. of Int. Conf. on Computer-Aided Design (ICCAD)*, 1993.

[43] J. CONG, K. S. LEUNG, and D. ZHOU. Performance-Driven Interconnect Design Based on Distributed RC Model. In *proc. on IEEE/ACM Design Automation Conference (DAC)*, 1993.

[44] J. CONG and Z. PAN. Interconnect Performance Estimation Models for Design Planning. *IEEE Trans. on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 20(6):739–752, 2001.

[45] M. COPPOLA, S. CURABA, M. GRAMMATIKAKIS, and G. MARUCCIA. IPSIM : SystemC 3.0 Enhancements for Communication Refinement. In *proc. of Design, Automation and Test in Europe (DATE)*, 2003.

[46] L. A. CORTES, P. ELES, and Z. PENG. Quasi-Static Assignment of Voltages and Optional Cycles for Maximizing Rewards in Real-time Systems with Energy Constraints. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2005.

[47] A. P. DANCY, R. AMIRTHARAJAH, and A. P. CHANDRAKASAN. High-Efficiency Multiple-Output DC-DC Conversion for Low-Voltage Systems. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 8(3):252–263, 2000.

[48] J. M. DAVEAU, T. B. ISMAIL, and A. A. JERRAYA. Synthesis of System-Level Communication by an Allocation-Based Approach. In *proc. Int. Symposium on System Synthesis*, 1995.

[49] J. M. DAVEAU, G. F. MARCHIORO, T. B. ISMAIL, and A. A. JERRAYA. Protocol Selection and Interface Generation for HW/SW Co-design. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 5(1)(No. 1):136–144, 1997.

[50] P. DE, E. DUNNE, J. GHOSH, and C. WELLS. Complexity of the Discrete Time-Cost Trade off Problem for Project Networks. *Operation research*, vol. 45(2):302–306, March 1997.

[51] R. H. DENNARD, F. H. GAENSSLEN, H.-N. YU, V. L. RIDEOUT, E. BASSOUS, and A. R. LEBLANC. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, SC-9(5):256–268, 1974.

[52] A. DHARCHOUDHURY and S. M. KING. Worst Case Analysis and Optimization of VLSI Circuit Performances. *IEEE Transaction of Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, Vol. 14(No. 4):481–492, 1995.

[53] J. DUATO, S. YALAMANCHILI, and L. NI. *Interconnection Networks*. Morgan Kaufmann Publishers, 2003.

[54] P. ELES, A. DOBOLI, P. POP, and Z. PENG. Scheduling with Bus Access Optimization for Distributed Embedded Systems. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 8(5):472–491, 2000.

[55] D. FLYNN. AMBA: Enabling Reusable On-Chip Designs. *IEEE Micro*, 17(4):20–27, 1997.

[56] W. FORNACIARI, D. SCIUTO, and C. SILVANO. Power Estimation for Architecture Exploration of HW/SW Communication on System Level Buses. In *proc. of CODES*, 1999.

[57] D. J. FRANK, P. SOLOMON, S. REYNOLDS, and J. SHIN. Supply and Threshold Voltage Optimization for Low Power Design. In *proc. of Int. Symposium on Low Power Electronics Design (ISLPED)*, 1997.

[58] J. FRENKIL. Tools and Methodologies for Low Power Design. In *proc. of Design Automation Conf. (DAC), Anaheim, California*, 1997.

[59] D. D. GAJSKI, N. DUTT, A. WU, and S. LIN. *High-Level Synthesis : Inroduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[60] D. D. GAJSKI, J. ZHU, R. DMER, A. GERSTLAUER, and S. ZHAO. *SpecC : Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[61] M. GASTEIER and M. GLESNER. Bus-Based Communication Synthesis on System-Level. In *proc. of Int. Symposium on System Synthesis*, 1996.

[62] M. GASTEIER and M. GLESNER. Bus-based Communication Synthesis on System Level. *ACM Trans. on Design Automation Electronic Systems (TODAES)*, 4(1):1–11, 1999.

[63] M. GASTEIER, M. MNCH, and M. GLESNER. Generation of Interconnect Topologies for Communication Synthesis. In *proc. of Design Automation and Test in Europe (DATE), Paris, France*, 1998.

[64] P. GELSINGER. Moore's Law - The Genius Lives On. *IEEE Solid-State Circuits Society Newsletter*, 20(3):18–20, 2006.

[65] G. GOGNIAT, M. AUGUIN, L. BIANCO, and A. PEGATOQUET. Communication Synthesis and Hw/Sw Integration for Embedded System Design. In *proc. of Int. Workshop on Hardware/Software Co-design, Seattle, Washington*, 1998.

[66] A. GOLDBERG and R. E. TARJAN. A New Approach to the Maximum Flow Problem. *Journal Assoc. comput. Mach.*, 35:921–940, 1988.

[67] R. GONZALEZ, B. M. GORDON, and M. A. HOROWITZ. Supply and Threshold Voltage Scaling for Low Power CMOS. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, 1997.

[68] J. GOODMAN, A. P. DANCY, and A. P. CHANDRAKASAN. An Energy/Security Scable Encryption Processor Using an Embedded variable Voltage DC/DC Converter. *IEEE Journal of Solid-State Circuits*, 33(11):1799–1809, 1998.

[69] K. G. W. GOOSSENS. A Protocol and Memory Manager for On-Chip Communication. In *proc. of IEEE Int. Symposium on Circuits and Systems*, 2001.

[70] D. M. GRANT and P. B. DENYER. Memory, Control and Communication Synthesis for Scheduled Algorithms. In *proc. of ACM/IEEE Design Automation Conference (DAC)*, 1990.

[71] F. GRUIAN and K. KUCHCINSKI. LEneS: Task Scheduling for Low Energy Systems Using Variable Supply Voltage Processors. In *proc. of Asia and South Pacific Design Automation Conference (ASPDAC)*, 2001.

[72] T. GRTKER, S. LIAO, G. MARTIN, and S. SWAN. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[73] M. HAMADA, M. TAKAHASHI, H. ARAKIDA, A. CHIBA, T. TERAZAWA, T. ISHIKAWA, M. KANAZAWA, M. IGARASHI, K. USAMI, and T. KURODA. A Top-Down Low Power Design Technique Using Clustered Voltage Scaling with Variable Supply-Voltage Scheme. In *proc. of IEEE Conf. on Custom Integrated Circuits*, 1998.

[74] J. HENKEL and R. ERNST. High-level estimation techniques for usage in hardware/software co-design,. In *proc. of Asia and South Pacific Design Automation Conference (ASPDAC)*, 1998.

[75] F. HESSEL, P. COSTE, G. NICOLESCU, P. LEMARREC, N. ZERGAINOH, and A. JERRAYA. Multilevel Communication Synthesis of Heterogeneous Multilanguage Specification. In *proc. of IEEE Conference*, 2000.

[76] K. HINES and G. BORRIELLO. Dynamic Communication Models in Embedded System Co-Simulation. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 1997.

[77] M. HIRABAYASHI, K. NOSE, and T. SAKURAI. Design Methodology and Optimization Strategy for Dual-Vth Scheme Using Commerrially Available Tools. In *proc. of Int. Symposium on Low Power Electronics and Design (ISLPED)*, 2001.

[78] D. HOMMAIS, F. PETROT, and I. AUGE. A Tool Box to Map System Level Communications on HW/SW Architectures. In *proc. of Int. Workshop on Rapid System Prototyping*, 2001.

[79] C. HSIEH and M. PEDRAM. Architectural Energy Optimization by Bus Splitting. *IEEE Trans. on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 21(4)(No. 4):408–414, April 2002.

[80] J. HU, Y. DENG, and R. MARCULESCU. System-Level Point-to-Point Communication Synthesis Using Floorplanning Information. In *proc. of Int. Conf. on VLSI Design (VLSID)*, 2002.

[81] F. ICHIBA, K. SUZUKI, S. MITA, T. KURODA, and T. FURUYAMA. Varriable Supply-Voltage Scheme with 95%-Efficiency DC-DC Converter for MPEG-4 Codec. In *proc. of Int Sypmosium Low Power Electronics Design (ISLPED)*, 1999.

[82] M. IGRASHI, K. USAMI, K. NOGAMI, F. MINAMI, Y. KAWASAKI, T. AOKI, M. TAKANO, C. MIZUNO, T. ISHIKAWA, M. KANAZAWA, S. SONODA, M. ICHIDA, and N. HATANAKA. A Low-Power Design Method Using Multiple Supply Voltages. In *proc. of Int Sypmosium Low Power Electronics Design (ISLPED)*, 1997.

[83] T. B. ISMAIL, M. ABID, and A. JERRAYA. COSMOS : A Co-design Approach for Communicating Systems. In *proc. of IEEE Int. Workshop on Hardware/Software Co-design*, 1994.

[84] H. F. JYU, S. MALIK, S. DEVADAS, and K. W. KEUTZER. Statistical Timing Analysis of Combinational Logic Circuits. *IEEE transactions on very large scale integration (VLSI) systems*, Vol. 1(No. 2):126–137, 1993.

[85] V. V. KAENEL, P. MACKEN, and M. G. R. DEGRAUWE. A Voltage Reduction Technique for Battery-Operated Systems. *IEEE Journal of Solid-State Circuits*, 25(5):1136–1140, 1990.

[86] P. KALL and S. W. WALLACE. *Stochastic Programming*. John wiley and sons, 1994.

[87] J. KAO, A. CHANDRAKASAN, and D. ANTONIADIS. Transistor Sizing Issues and Tool for Multi-Threshold CMOS Technology. In *proc. of Design Automation Conf. (DAC), Anaheim, California*, 1997.

[88] F. KARIM, A. NGUYEN, S. DEY, and R. RAO. On-Chip Communication Architecture for OC-768 Network Processors. In *proc. of Design Automation Conference (DAC), Las Vegas, Nevada*, 2001.

[89] G. KHAN. The Semantics of a Simple Language for Parallel Programming. In *proc. IFIP Congress 74, North-Holland, Amsterdam*, 1974.

[90] B. W. KIM and C. M. KYUNG. Exploiting Intellectual Properties With Imprecise Design Costs for System-on-Chip Synthesis. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 10(3):240–252, 2002.

[91] P. V. KNUDSEN and J. MADSEN. Integrating Communication Protocol Selection with Hardware/Software Codesign. *IEEE Trans. on Comupter-Aided Design (TCAD) of Integrated Circuits and Systems*, 18(8):1077–1095, 1999.

[92] T. KOLKS, B. LIN, and H. D. MAN. Sizing and Verification of Communication Buffers for Communicating Processes. In *proc. of Int. Conference on Computer-Aided Design*, 1993.

[93] H. KOPETZ, A. DAMM, C. KOZA, M. MULAZZANI, W. SCHWABL, C. SENFT, and R. ZAINLINGER. Distributed Fault-Tolerant Real-time Systems: the Mars Approach. *IEEE Micro*, 9(1):25–40, 1989.

[94] H. KOPETZ and G. GRNSTEIDL. TTP-A Protocol for Fault Tollerent Real-Time Systems. *IEEE Computer*, 27(1):14–23, 1994.

[95] K. LAHIRI and A. RAGHUNATHAN. Power Analysis of System-Level On-Chip Communication Architectures. In *proc. Int. Conf. on Hardware/Software Co-design and System Synthesis (CODES+ISSS), Stockholm, Sweden*, 2004.

[96] K. LAHIRI, A. RAGHUNATHAN, and S. DEY. Design Space Exploration for Optimizing On-Chip Communication Architecture. *IEEE Trans. on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 23(6)(No. 6):952–961, June 2004.

[97] J. LEHOCZKY, L. SHA, and Y. DING. The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior. In *proc. of IEEE Real-Time Systems Symposium*, 1989.

[98] C. E. LEISERSON and J. B. SAXE. Retiming Synchronous Circuitry. *Algorithmica, Springer New York*, 6(1):5–35, 1991.

[99] C. K. LENNARD, P. SCHAUMONT, G. D. JONG, A. HAVERINEN, and P. HARDEE. Standards for System-Level Design: Practical Reality or Solution in Search of a Question? In *proc. of Design, Automation and Test in Europe (DATE)*, 2000.

[100] J. LI and M. CHEN. Generating Explict Communication from Shared-Memory Program References. In *proc. of IEEE Conference*, 1990.

[101] P. LIEVERSE, P. V. WOLF, and E. DEPRETTERE. A Trace Transformation Technique for Communication Refinement. In *proc. of Int. Symposium on Hardware/Software Co-design*, 2001.

[102] J. LILLIS, C.-K. CHENG, and T.-T. Y. LIN. Optimal and Efficient Buffer Insertion and Wire Sizing. In *proc. of Int. Conf. on Custom Integrated Circuits*, 1995.

[103] J. J. LIOU, K. T. CHENG, S. KUNDU, and A. KRSTIC. Fast Statistical Timing Analysis by Probability Event Propagation. In *proc. of Design Automation Conference (DAC)*, 2001.

[104] D. L. LIU and C. SVENSSON. Power Consumption Estimation in CMOS VLSI chips. *IEEE Journal SSC*, vol. 29(6):1531–1549, June 1994.

[105] D. LYONNARD, S. YOO, A. BAGHDADI, and A. A. JERRAYA. Automatic Generation of Application Specific Architectures for Heterogeneous Multiprocessor SoC. In *proc. of Design Automation Conference (DAC)*, 2001.

[106] M. MANI, A. DEVGAN, and M. ORSHANSKY. An Efficient Algorithm for Statistical Minimization of Total Power Under Timing Yield Constraints. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2005.

[107] S. MARTIN, K. FLAUTNER, T. MUDGE, and D. BLAAUW. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Low Power Microprocessor Under Dynamic Workloads. In *proc. of Int. Conf. on Computer Aided Design (ICCAD)*, pages 721–725, 2002.

[108] J. MEINDL. Low Power Microelectronics: Retrospect and Prospect. *Proc. of IEEE*, 83(4):619–635, 1995.

[109] P. MICHEL, U. LAUTHER, and P. DUZY. *The Synthesis Approach to Digital System Design*. Kluwer Academic Publishers, 1992.

[110] G. MOORE. Cramming More Components onto Integrated Circuit. *Electronics*, Vol. 38(No. 8):114–117, 1965.

[111] T. MURGAN, M. MOMENI, A. G. ORTIZ, and M. GLESNER. A High-Level Compact Pattern-Dependent Delay Model for High-Speed Point-to-point Interconnects. In *proc. of Int. Conference on Computer Aided Design (ICCAD)*, 2006.

[112] T. A. MURGAN. *High-Level Optimization of Peformance and Power in Very Deep Sub-Micron Interconnects*. PhD thesis, Darmstadt University of Technology, Germany, 2006.

[113] J. MUTTERSBACH, T. VILLIGER, H. KAESLIN, N. FELBER, and W. FICHTNER. Globally Asynchronous Locally Synchronous Architectures to Simplify the Design of On-Chip Systems. In *proc. of IEEE Int. Conf. on ASIC/SoC*, 1999.

[114] S. NARAYAN and D. D. GAJSKI. Synthesis of System-Level Bus Interfaces. In *proc. of Design Automation and Test in Europe (DATE)*, 1994.

[115] S. NASSIF. Delay Variability: Sources, Impacts and Trends. In *proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, 2000.

[116] C. NEAU and K. ROY. Optimal Body Bias Selection for Leakage Improvement and Process Compensation Over Different Technology Generations. In *proc. of Int. Symposium on Low Power Electronics Design (ISLPED)*, 2003.

[117] Y. NESTEROV and A. NEMIROVSKII. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied Mathematics, 1994.

[118] G. NICOLESCU, S. YOO, and A. JERRAYA. Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design. In *proc. of Design, Automation and Test in Europe (DATE)*, 2001.

[119] O. OGAWA, S. B. DE NOYER, P. CHAUVET, K. SHINOHARA, Y. WATANABE, H. NIIZUMA, T. SASAKI, and Y. TAKAI. A Practical Approach for Bus Architecture Optimization at Transaction Level. In *proc. of Design, Automation and Test in Europe (DATE)*, 2003.

[120] T. OKUMA, H. YASUURA, and T. ISHIHARA. Software Energy Reduction Techniques for Variable Voltage Processors. *IEEE Design and Test of computers*, 18 (2):31–41, March 2001.

[121] M. ORSHANSKY, J. C. CHEN, and C. HU. Direct Sampling Methodology for Statistical Analysis of Scaled CMOS Technologies. *In IEEE Transactions on semiconductor manufacturing*, Vol. 12(No. 4):403–408, 1999.

[122] R. B. ORTEGA and G. BORRIELLO. Communication Synthesis for Embedded Systems with Global Considerations. In *proc. of Int. Workshop on Hardware/Software Co-design (CODES/CASHE)*, 1997.

[123] R. B. ORTEGA and G. BORRIELLO. Communication Synthesis for Distributed Embedded Systems. In *proc. of Int. Conference on Computer-Aided Design (ICCAD), San Jose, California*, 1998.

[124] R. H. J. M. OTTEN. Automatic Floorplan Design. In *proc. of ACM/IEEE Design Automation Conference (DAC)*, 1982.

[125] A. PAPOULIS and S. U. PILLAI. *Probability, Random Varibles, and Stochastic Processes*. Mc Graw Hill, fourth edition.

[126] S. PASRICHA, N. DUTT, and M. BEN-ROMDHANE. Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration. In *proc. of Design Automation Conference (DAC), San Diego, California*, 2004.

[127] S. PASRICHA, N. DUTT, E. BOZORGZADEH, and M. BEN-ROMDHANE. Floorplan Aware Automated Synthesis of Bus-Based Communication Architectures. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2005.

[128] S. PASRICHA, N. DUTT, E. BOZORGZADEH, and M. BEN-ROMDHANE. FABSYN: Floorplan-Aware Bus Architecture Synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, Vol. 14(No. 3):241–253, 2006.

[129] S. PHILLIPS and M. I. DESSOUKY. Solving the Project Time/Cost Trade-off Problem Using the Minimal Cut Concept. *Management Science*, Vol. 24:393–400.

[130] A. D. PIMENTEL and C. ERBAS. An IDF-based Trace Transformation Method for Communication Refinement. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2003.

[131] A. D. PIMENTEL, L. O. HERTZBERGER, P. LIEVERSE, P. V. WOLF, and E. F. DEPRETTERE. Exploring Embedded-Systems Architectures with Artemis. *Computer*, 18(1):57–63, 2001.

[132] A. PINTO, L. P. CARLONI, and A. V. SANGIOVANNI. Constraint Driven Communication Synthesis. In *proc. of Design Automation Conference (DAC), New Orleans, Louisiana*, June 2002.

[133] P. POP, P. ELES, and Z. PENG. Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems. In *proc. of Int. Conf. on Real-Time Computing Systems and Applications (RTCSA)*, 1999.

[134] V. RAGHUNATHAN, M. B. SRIVASTAVA, and R. K. GUPTA. A Survey of Techniques for Energy Efficient On-Chip Communication. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2003.

[135] S. RAJ, S. B. K. VRUDHULA, and J. WANG. A methodology to improve timing yield in the presence of process variations. In *proc. of Design Automation Conference (DAC)*, 2004.

[136] J. A. RICE. *Mathematical Statistics and Data Analysis*. Second edition, Duxbury press, 1995.

[137] K. V. ROMPAEY, D. VERKEST, I. BOLSENS, and H. D. MAN. CoWare - A Design Environement for Heterogeneous Hardware/Software Systems. *IEEE Design Automation Embedded Systems*, 1(4):357–386, 1996.

[138] K. K. RYE and V. MOONEYIII. Automated Bus Generation for Multiprocessor SoC Design. *IEEE Trans. on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 23(11)(No. 11):1531–1549, Nov. 2004.

[139] G. A. SAI-HALASZ. Performance Trends in High-Performane Processors. In *proc. of IEEE*, 1995.

[140] S. B. SAMAAN. The Impact of Device Parameters Variations on the Frequency and Performance of VLSI Chips. In *proc. of Int. Conf. on Computer-Aided Design (ICCAD)*, 2004.

[141] S. S. SAPATNEKAR. RC Interconnect Optimization Under the Elmore Delay Model. In *proc. of IEEE/ACM Design Automation Conference (DAC)*, 1994.

[142] A. SCHRIJVER. *Theory of Linear and Interger Programming*. John Wiley & Sons, 1986.

[143] K. SEKAR, K. LAHIRI, A. RAGHUNATHAN, and S. DEY. FLEXBUS: A High Performance System-on-Chip Communication Architecture with a Dynamically Configurable Topology. In *proc. of Design Automation Conference (DAC), Anaheim, California*, 2005.

[144] A. SIEBENBORN, O. BRINGMANN, and W. ROSENSTIEL. Communication Analysis for System on Chip Design. In *proc. of Design, Automation and Test in Europe (DATE)*, 2004.

[145] M. SKUTELLA. Approximation Algorithms for the Discrete Time-Cost Trade-off Problem. *Mathematics of operation research*, vol. 23(4):909–929, Nov. 1998.

[146] A. SRIVASTAVA and D. SYLVESTER. A General Framework for Probabilistic Low Power Design Space Exploration Considering Process Variation. In *proc. of Int. Conference on Computer-Aided Design (ICCAD)*, 2004.

[147] A. SRIVASTAVA, D. SYLVESTER, and D. BLAAUW. *Statistical Analysis and Optimization of VLSI : Timing and Power*. Springer Science + Business Media, Inc., 2005.

[148] M. R. STAN and W. P. BURLESON. Bus-Invert Coding for Low-Power I/O. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 3(1):49–58, 1995.

[149] M. R. STAN and W. P. BURLESON. Low-Power Encodings for Global Communication in CMOS VLSI. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 5(4):444–455, 1997.

[150] B. SVANTESSON, S. KUMAR, and A. HEMANI. A Methodology and Algorithms for Efficient Interprocess Communication Synthesis from System Description in SDL. In *proc. of Int. Conf. on VLSI Design (VLSID)*, 1997.

[151] C. SVENNSON. *"Low Voltage Technologies", Low Power Design in Deep Submicron Electronics*. Kluwer Academic Publishers, Dordrecht, 1997.

[152] R. SWANSSON and J. MEINDL. Ion-Implanted Complementary MOS Transistors in Low Voltage Circuits. *IEEE Journal on Solid-State Circuits (JSS)*, 7:146–153, 1972.

[153] D. SYLVESTER and K. KEUTZER. Getting to the Bottom of Deep Sub-micron. In *proc. of Int. Conf. on Computer-Aided Design (ICCAD)*, 1998.

[154] D. SYLVESTER and K. KEUTZER. Impact of Small Process Geometries on Microarchitectures in Systems on a Chip. In *proc. of the IEEE*, 2001.

[155] Y. TAUR and E. NOWAK. CMOS devices below 0.1 μm: How high will performance go? *IEDM Technical Digest*, pages 215–218, 1997.

[156] N. THEPAYASUWAN and A. DOBOLI. Layout Conscious Bus Architecture Synthesis for Deep Submicron Systems-on-Chip. In *proc. of the Design, Automation and Test in Europe (DATE)*, 2004.

[157] N. THEPAYASUWAN and A. DOBOLI. Layout Conscious Approach and Bus Architecture Synthesis for Hardware/Software Co-design of Systems-on-Chip Optimizied for Speed. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 13(5):525–538, 2005.

[158] H. J. TOUATI. *Performance-Oriented Technology Mapping*. PhD thesis, Dept. of Electrical Engineering and Copmuter Science, UC Berkeley, Nov. 1990.

[159] Y.-F. TSAI, D. DUARTE, N. VIJAYAKRISHNAN, and M.-J. IRWIN. Impact of Process Scaling on the Efficacy of Leakage Reduction Schemes. In *proc. of Int. Conf. on Integrated Circuit Design and Technology*, 2004.

[160] J.-C. TSAY and Y.-C. HO. High-Level Synthesis of Shared-Bus Systems from Data-Flow Graphs. *Proceedings of the National Science Council, Republic of China*, 23(1):133–142, 1999.

[161] C. J. TSENG and D. P. SIEWIOREK. Automated Synthesis of Data Paths on Digital Systems. *IEEE Transaction of Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, Vol. CAD-5(No. 3):397–395, 1986.

[162] E. D. V. R. YON KAENEL, M. D. PARDOEN and E. A. VITTOZ. Automatic Adjustment of Threshold and Supply Voltages for Minimum Power Consumption in CMOS Digital Circuits. In *proc. of IEEE Symposium on Low Power Electronics Design (ISLPED)*, 1994.

[163] F. VAHID and L. TAURO. An Object-Oriented Communication Library for Hardware-Software Co-design. In *proc. of Int. Workshop on Hardware/Software Co-design (CODES/CASHE)*, 1997.

[164] L. P. P. P. VAN GINNEKEN. Buffer Placement in Distributed RC-Tree Networks for Minimal Elmore Delay. In *proc. of Int. Symposium on Circuit and Systems*, 1990.

[165] G. VARATKAR and R. MARCULESCU. On-Chip Communication Analysis for Multimedia Applications. In *proc. of IEEE Int. Conf. on Multimedia and Expo (ICME)*, 2002.

[166] G. V. VARATKAR and R. MARCULESCU. On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 12(1):108–119, 2004.

[167] N. H. E. WESTE and K. ESHRAGHIAN. *Principles of CMOS VLSI Design*. Addison wesley, 1994.

[168] D. F. WONG and C. L. LIU. A New Algorithm for Floorplan Design. In *proc. of ACM/IEEE Design Automation Conference (DAC)*, 1986.

[169] L. YAN, J. LUO, and N. JHA. Joint Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems. *IEEE Transaction on Computer-Aided Design (TCAD) of Integrated Circuits and Systems*, 24(7):1030–1041, Nov. 2005.

[170] T. Y. YEN and W. WOLF. Communication Synthesis for Distributed Embedded Systems. In *proc. of Int. Conference on Computer Aided Design (ICCAD)*, 1995.

[171] K. Y. YUN and R. P. DONOHUE. Pausible Clocking: A First Step Toward Heterogenous Systems. In *proc. of Int. Conf. Computer Design (ICCD)*, 1996.

[172] X. ZHU and S. MALIK. A Hierarchical Modeling Framework for On-Chip Communication Architectures. In *proc. of Int. Cont. on Comuputer-Aided Design (ICCAD)*, 2002.

[173] P. ZUCHOWSKI, P. A. HABITZ, J. D. HAYES, and J. H. OPPOLD. Process and Environment Variation Impacts on ASIC Timing. In *proc. of Int. Cont. on Computer-Aided Design (ICCAD)*, 2004.

List of Publications

[174] S. PANDEY and M. GLESNER. Simultaneous On-Chip Bus Synthesis and Voltage Scaling Under Random On-Chip Data Traffic. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 2007 (Accepted for Publication).

[175] S. PANDEY, N. UTLU, and M. GLESNER. Tabu Search Based On-Chip Communication Bus Synthesis for Shared Multi-Bus Based Architecture. In *proc. of* 14^{th} *IFIP Int. Conf. on VLSI-SoC, Nice, France*, 2006.

[176] S. PANDEY, T. MURGAN, and M. GLESNER. Energy Conscious Simultaneous Voltage Scaling and On-Chip Communication Bus Synthesis. In *proc. of* 14^{th} *IFIP Int. Conf. on VLSI-SoC, Nice, France*, 2006.

[177] S. PANDEY and M. GLESNER. Energy Efficient Statistical On-Chip Communication Bus Synthesis for Reconfigurable Architecture. In *proc. of* 16^{th} *IEEE Int. Conf. on Field Programmable Logic Application (FPL), Madrid, Spain*, 2006.

[178] S. PANDEY and M. GLESNER. Statistical On-Chip Communication Bus Synthesis and Voltage Scaling Under Timing Yield Constraint. In *proc. of IEEE/ACM Design Automation Conference (DAC), San Francisco*, 2006.

[179] S. PANDEY and M. GLESNER. Energy Efficient MPSoC On-Chip Communication Bus Synthesis Using Voltage Scaling Technique. In *proc. of IEEE Int. Symposium on Circuit and System (ISCAS)*, 2006.

[180] S. PANDEY, M. GLESNER, and M. MÜHLHÄUSER. Performance Aware On-Chip Communication Synthesis and Optimization for Shared Multi-Bus Based Architecture. In *proc. of* 18^{th} *ACM Symposium on Circuit and System (SBCCI), Florianopolis, Brazil*, 2005.

[181] S. PANDEY, M. GLESNER, and M. MÜHLHÄUSER. On-Chip Communication Topology Synthesis for Shared Multi-Bus Based Architecture. In *proc. of* 15^{th} *IEEE Int. Conf. on Field Programmable Logic Application (FPL), Tampere, Finland*, 2005.

[182] S. PANDEY, M. GLESNER, and M. MÜHLHÄUSER. Architecture Level Design Space Exploration and Mapping of Hardware. In *proc. of IEEE Int. Symposium on Signal, Circuits and Systems (ISSCS)*, 2005.

[183] S. PANDEY, M. GLESNER, and M. MÜHLHÄUSER. High Level Hardware/Software Communication Estimation in Shared Memory Architecture. In *proc. of IEEE Int. Symposium on Circuits and Systems (ISCAS), Kobe, Japan*, 2005.

[184] S. PANDEY, P. ZIPF, O. SOFFKE, and M. GLESNER. Ubicomp Device for a Decentralized Distributed Computing Environment. In *proc. of IEEE Int. Symposium on Signal Processing and Information Technology (ISSPIT), Darmstadt, Germany*, 2003.

[185] S. PANDEY, P. ZIPF, O. SOFFKE, M. PETROV, T. MURGAN, M. GLESNER, and M. MÜHLHÄUSER. An Infrastructure for Distributed Computing and Context Aware Computing. In *Workshop: Multi-Device Interfaces for Ubiquitous Peripheral Interaction, at the Fifth Int. Conf. on Ubiquitous Computing, Seattle, Washington, USA*, 2003.

[186] T. MURGAN, P. BACINSCHI, S. PANDEY, A. GARCIA ORTIZ, and M. GLESNER. On the Necessity of Combining Coding with Spacing and Shielding for Improving Performance and Power in Very Deep Sub-micron Interconnects. In *proc. of Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Göteborg, Sweden*, 2007.

[187] T. MURGAN, O. MITEA, S. PANDEY, P. BACINSCHI, and M. GLESNER. Simultaneous Placement and Buffer Planning for Reduction of Power Consumption in Interconnects and Repeaters. In *proc. of* 14^{th} *IFIP Int. Conf. on VLSI-SoC, Nice, France*, 2006.

[188] T. HOLLSTEIN, S. PANDEY, and M. GLESNER. Advanced On-Chip Communication Architecture and Routing Methods for System-on-Chip. In *European Workshop on Reconfigurable Communication Centric SoCs (ReCoSoC), Montpellier, France*, 2005.

[189] M. GLESNER, T. MURGAN, L. S. INDRUSIAK, M. PETROV, and S. PANDEY. System Design and Integration in Pervasive Appliances. *Journal of Microelectronics, Electronic Components and Materials (MIDEM)*, 4, 2003.

[190] M. GLESNER, T. MURGAN, L. S. INDRUSIAK, M. PETROV, and S. PANDEY. System Design and Integration in Pervasive Appliances. In *Proc. of the Int. Conf. on Microelectronics, Devices and Materials, Ptuj, Slovenia*, 2003.

Supervised Theses

[191] O. MITEA. Power Optimized Buffer Insertion in Very Deep Sub-Micron Technologies. Bachelor's thesis, Institute of Microelectronics Systems, Darmstadt University of Technology, Darmstadt, Germany, 2006.

[192] N. UTLU. Tabu Search Based On-Chip Communication Synthesis for Shared-Bus-Architecture. Bachelor's thesis, Institute of Microelectronics Systems, Darmstadt University of Technology, Darmstadt, Germany, 2005.

[193] Z. MING. Architecture Exploration for Speech-Feature-Extraction Acceleration. Bachelor's thesis, Institute of Microelectronics Systems, Darmstadt University of Technology, Darmstadt, Germany, 2005.

[194] G. MASALSKIS AND I BULOTAITE. Ogg Vorbis Decoder Implementation Analysis. Technical Report, Institute of Microelectronics Systems, Darmstadt University of Technology, Darmstadt, Germany, 2005.

[195] P. R. VAJRAVELU. Design and Optimization of Leon-Based FFT Co-Processor. Master's thesis, Institute of Microelectronic Systems, Darmstadt University of Technology, Darmstadt, Germany, 2005.

[196] L. WEIDONG. Development and Optimization of an Embedded AMBA Master for Compact Flash Interfacing. Master's thesis, Institute of Microelectronics Systems, Darmstadt University of Technology, Darmstadt, Germany, 2004.

Curriculum Vitae

Sujan PANDEY

Personal Data:

Date of Birth:	21. October 1976
Place of Birth:	Kathmandu, Nepal

Academic Degrees and Awards:

1987 - 1991	Visited middle and high school at "Vijaya Memorial High School", Kathmandu, Nepal Degree: School Leaving Certificate (S.L.C)
1992 - 1994	Patan Multiple Campus, Kathmandu, Nepal Degree: Intermediate in Science, I.Sc. (10+2)
1995 - 1999	Undergraduate student at the department of Electrical and Electronics Engineering, Kathmandu University, Nepal Degree: Bachelor in Engineering (B.E.)
2000 - 2002	Graduate student at University of Applied Sciences Offenburg, Germany Degree: Master of Science (M.Sc.)
2003 - 2007	Ph.D. student at the Institute of Microelectronic Systems, Darmstadt University of Technology, Germany
2000 - 2002	Graduate scholarship award for M.Sc. program from German Academic Exchange Service (DAAD)
2003 - 2005	Graduate research scholarship award for "System Integration of Ubiquitous Computing in Information Technology" funded by German Research Foundation (DFG)